Coinductive Program Verification

Thesis Proposal

Brandon Moore

University of Illinois

December 12, 2013

Outline

Goal

Program Verification for every language

Goal

Program Verification from (multi-step) Operational Semantics

- Always need executable semantics, to test formalization
- Can we avoid axiomatic semantics?
- Why operational?
  - Denotational is a whole different story
  - Don't know how to handle big step

Goal

Program Verification from (multi-step) Operational Semantics

- Always need executable semantics, to test formalization
- Can we avoid axiomatic semantics?
- Why operational?
  - Denotational is a whole different story
  - Don't know how to handle big step

Goal

Program Verification from (multi-step) Operational Semantics

- Always need executable semantics, to test formalization
- Can we avoid axiomatic semantics?
- Why operational?
  - Denotational is a whole different story
  - Don't know how to handle big step

Program verifiers should be at least *certifying*

Certifying

A *certifying* verifier produces a proof certificate along with claims

Certified

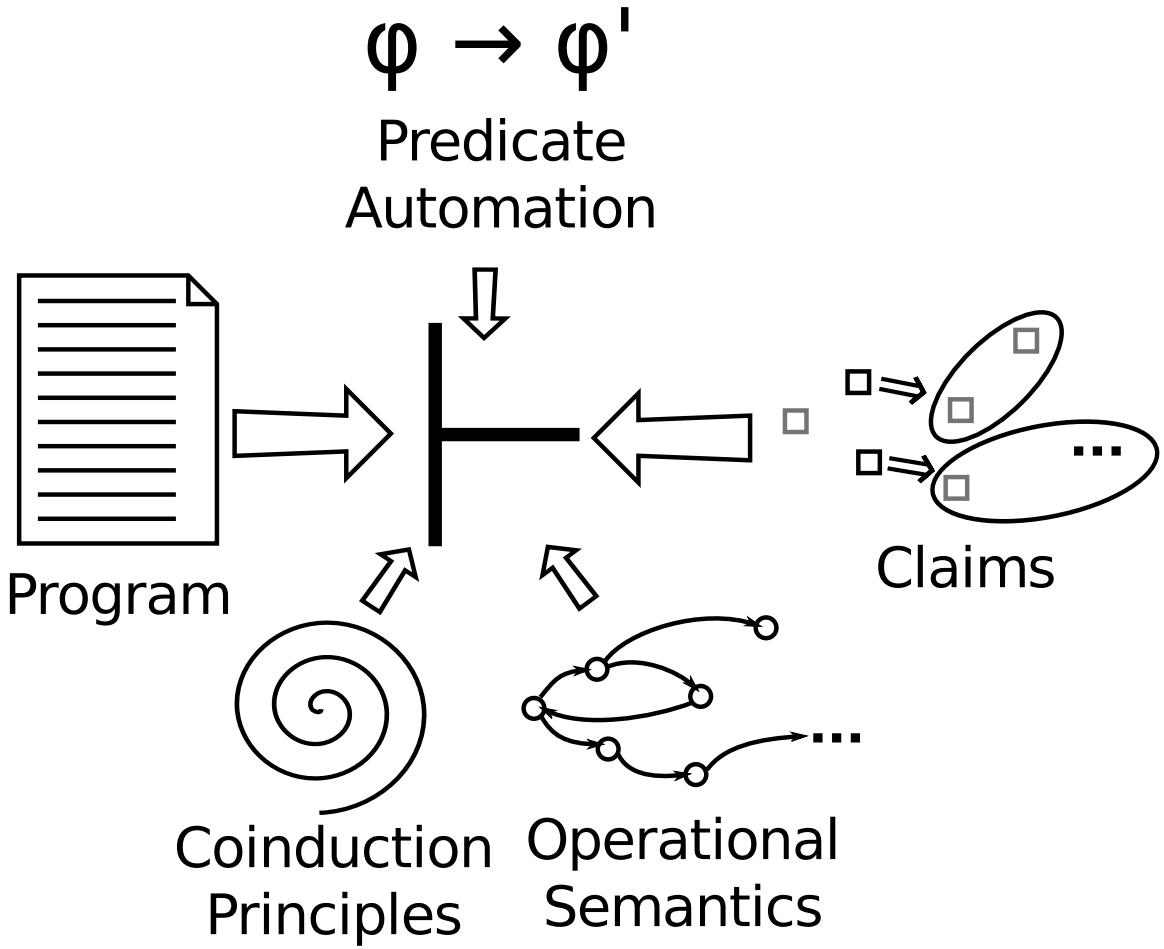
A *certified* verifier has a proof that it returns only true claims.

- Semantics in certificate language
- Translate specifications to claims about semantics
- Certificates are proofs claims are true
- Coq for certificate language, proof checker

General Perspective

- Language independence by passing from syntax to extension
  - Semantics, specifications, proof principles, etc.
- Truth/Proof as inclusion
- Coinduction

Project Structure



Operational semantics

Definition

An *Operational Semantics* is a set cfg of configurations and a one-step transition relation $S \subseteq cfg \times cfg$

A simple imperative language

- $cfg = Stmt \times (Var \rightharpoonup \mathbb{Z})$, written $\langle code, store \rangle$
- S contains steps like

$\langle$`while(n!=0) {s=s+n; n=n-1}`,
$\{$`s`$\mapsto 1$,`n`$\mapsto 10\}\rangle$

to

$\langle$`s=s+n; n=n-1; while(n!=0) {s=s+n; n=n-1}`,
$\{$`s`$\mapsto 1$,`n`$\mapsto 10\}\rangle$

Multiple Steps

Transitive closure S^* takes multiple steps, e.g.

$$\langle \texttt{while(n!=0) \{s=s+n; n=n-1\}; x=s}, \{\texttt{s} \mapsto 1, \texttt{n} \mapsto 10, \texttt{x} \mapsto 0\} \rangle$$

to

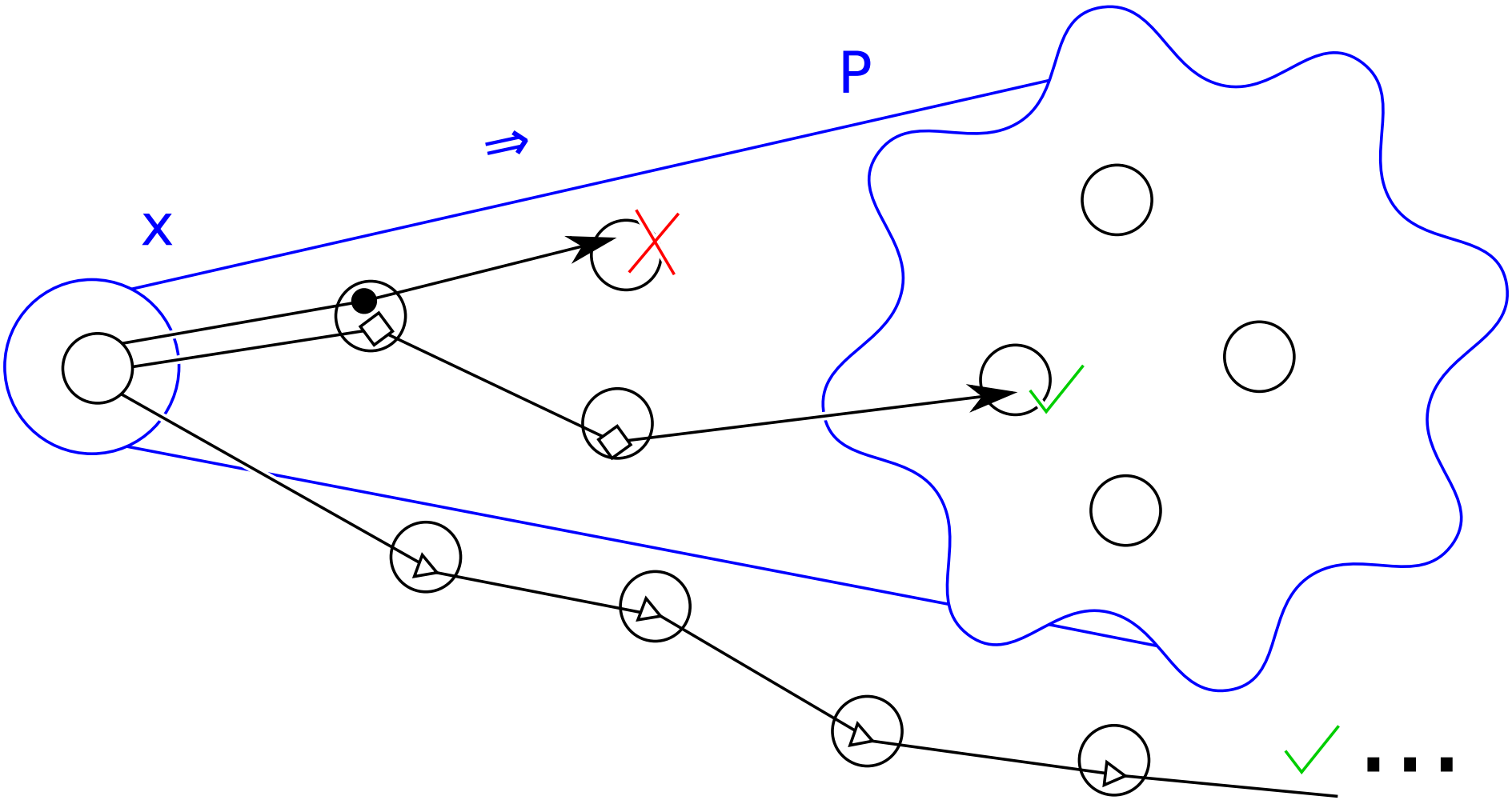
$$\langle \texttt{x = s}, \{\texttt{s} \mapsto 56, \texttt{n} \mapsto 10, \texttt{x} \mapsto 1\} \rangle$$

Reachability

Definition

A configuration $x \in cfg$ *"reaches"* a set $P \subseteq cfg$ of configurations, written $x \Rightarrow P$, when $x\, S^*\, y$ for some $y \in P$, or x diverges in S

Reachability in Pictures



Reachability from Hoare Triples

Start with a standard Hoare Triple

$\{\mathtt{s} = s_0, \mathtt{n} = n_0\}$
`while(n!=0) {s=s+n; n=n-1}`
$\{\mathtt{s} = s_0 + \sum_{i=0}^{n_0} i\}$

Drop special syntax for variables

Ordinary predicates on *store* of configuration

$\{store(\texttt{s}) = s_0 \wedge store(\texttt{n}) = n_0\}$
`while(n!=0) {s=s+n; n=n-1}`
$\{store(\texttt{s}) = s_0 + \sum_{i=0}^{n_0} i\}$

Drop special role of code

Ordinary predicates on configuration γ

$$\{\gamma.store(\texttt{s}) = s_0 \wedge \gamma.store(\texttt{n}) = n_0 \\ \wedge \gamma.code = \texttt{while(n!=0) \{s=s+n; n=n-1\};} R\}$$

$$\{\gamma.store(\texttt{s}) = s_0 + \sum_{i=0}^{n_0} i \\ \wedge \gamma.code = R\}$$

Matching Logic Reachability

Spec became a matching logic reachability property:

$$\varphi \Rightarrow_{RL} \varphi'$$

(φ, φ' predicates on *cfg*)

Back to Basic Reachability

Expands to reachability claims

$$\varphi \Rightarrow_{RL} \varphi'$$

iff

$$\forall \gamma, \forall \overline{fv(\varphi)}, \varphi(\gamma) \rightarrow \left(\gamma \Rightarrow \{\gamma' \mid \exists \overline{fv(\varphi') \setminus fv(\varphi)}, \varphi'(\gamma')\}\right)$$

Approach

- Specifications become sets of claims
- Proving sets of claims
- Coinduction
- Derived rules

Truth as Set

Definition

Let $claims = cfg \times \mathcal{P}(cfg)$, naming the set of all reachability claims

Definition

Let $reaches \subseteq claims$ be the set of true reachability claims,

$$reaches = \{(x, P) \mid x \text{ reaches } P\}$$

Proof by Inclusion

$$x \Rightarrow P \text{ for all } (x, P) \in R$$

iff

$$R \subseteq \mathit{reaches}$$

Inclusion by Coinduction

Definition

Given set A and function $F : \mathcal{P}(A) \to \mathcal{P}(A)$, $X \subseteq A$ is *F-stable* if $X \subseteq F(X)$

Coinduction

If G is the greatest fixpoint of a monotone F, $X \subseteq F(X)$ implies $X \subseteq G$

Reachability as a Fixpoint

reaches is the greatest fixpoint of $step : \mathcal{P}(claims) \to \mathcal{P}(claims)$

$$step[R] = done \cup next[R]$$

where $done : \mathcal{P}(claims)$, $next : \mathcal{P}(claims) \to \mathcal{P}(claims)$

$$done = \{(x, P) \mid x \in P\}$$
$$next[R] = \{(x, P) \mid \exists y.x\ S\ y \wedge (y, P) \in R\}$$

(Proof)

done in pictures

$(a, P) \in \mathit{done}$

$(b, P) \in \mathit{done}$

$(c, P) \in \mathit{done}$

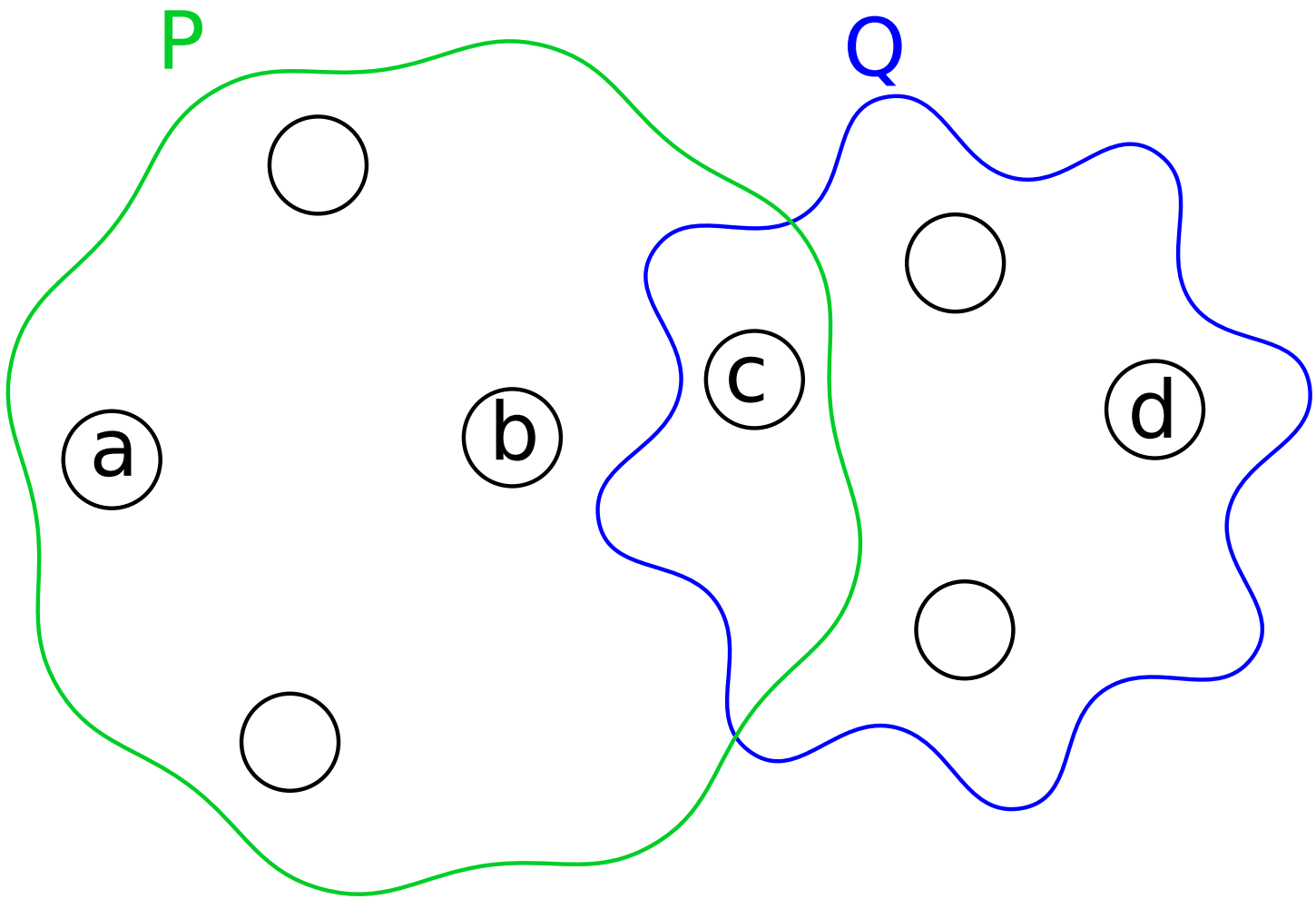
$(d, P) \notin \mathit{done}$

$(a, Q) \notin \mathit{done}$

$(b, Q) \notin \mathit{done}$

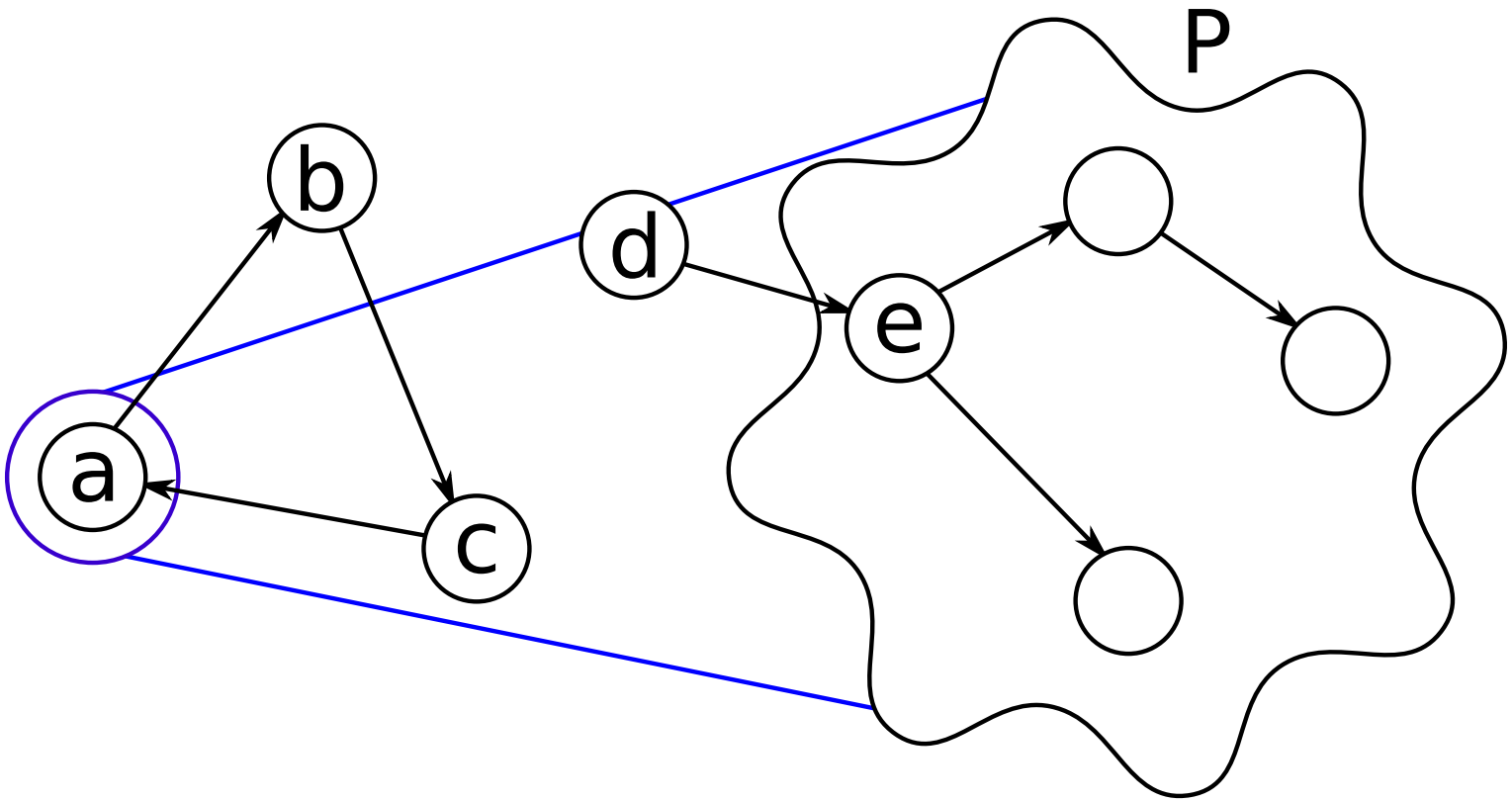
$(c, Q) \in \mathit{done}$

$(d, Q) \in \mathit{done}$



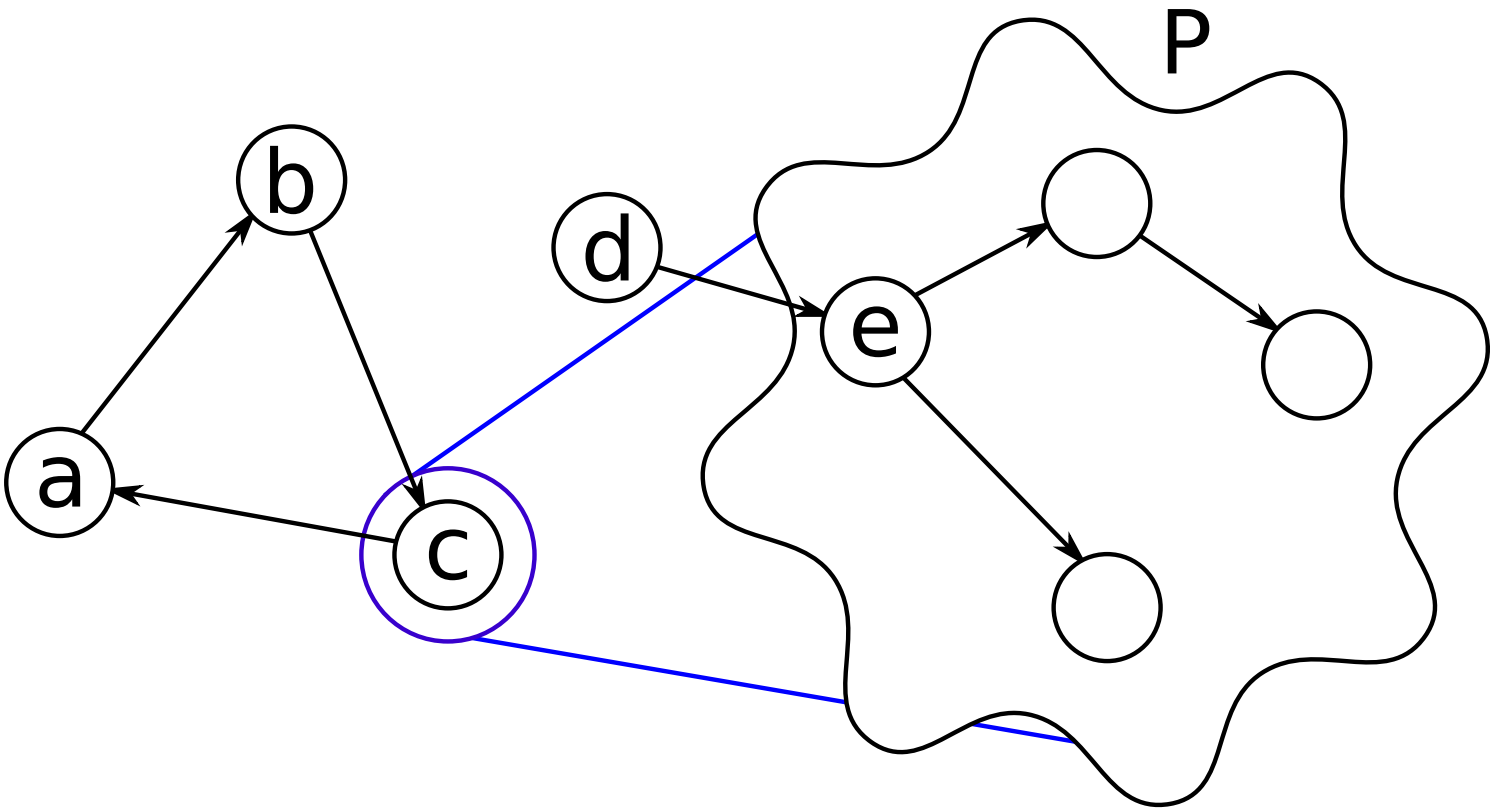
next in pictures

R



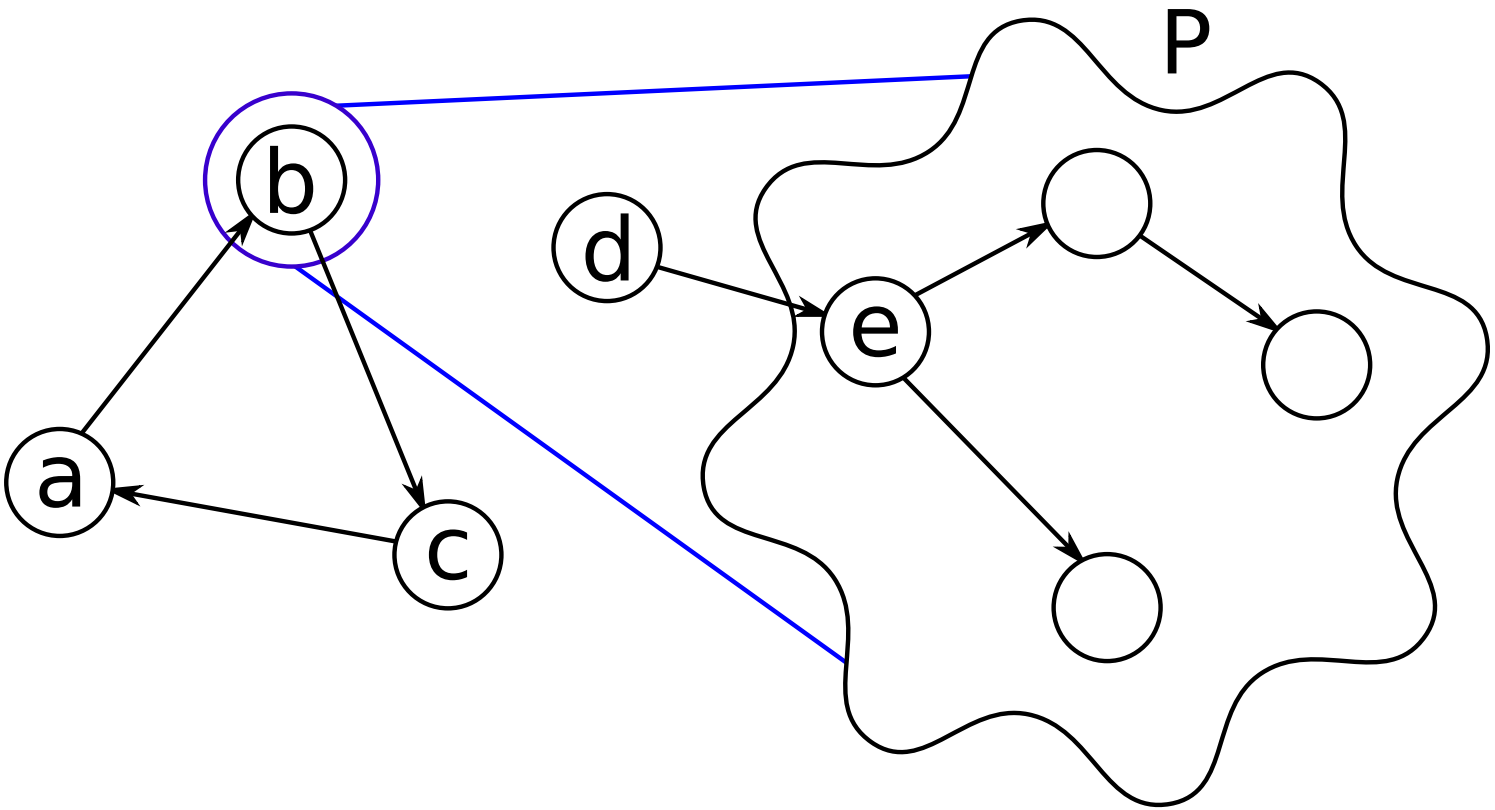
next in pictures

$next[R]$



next in pictures

$next[next[R]]$



Reachability as a Fixpoint

reaches is the greatest fixpoint of $step : \mathcal{P}(claims) \to \mathcal{P}(claims)$

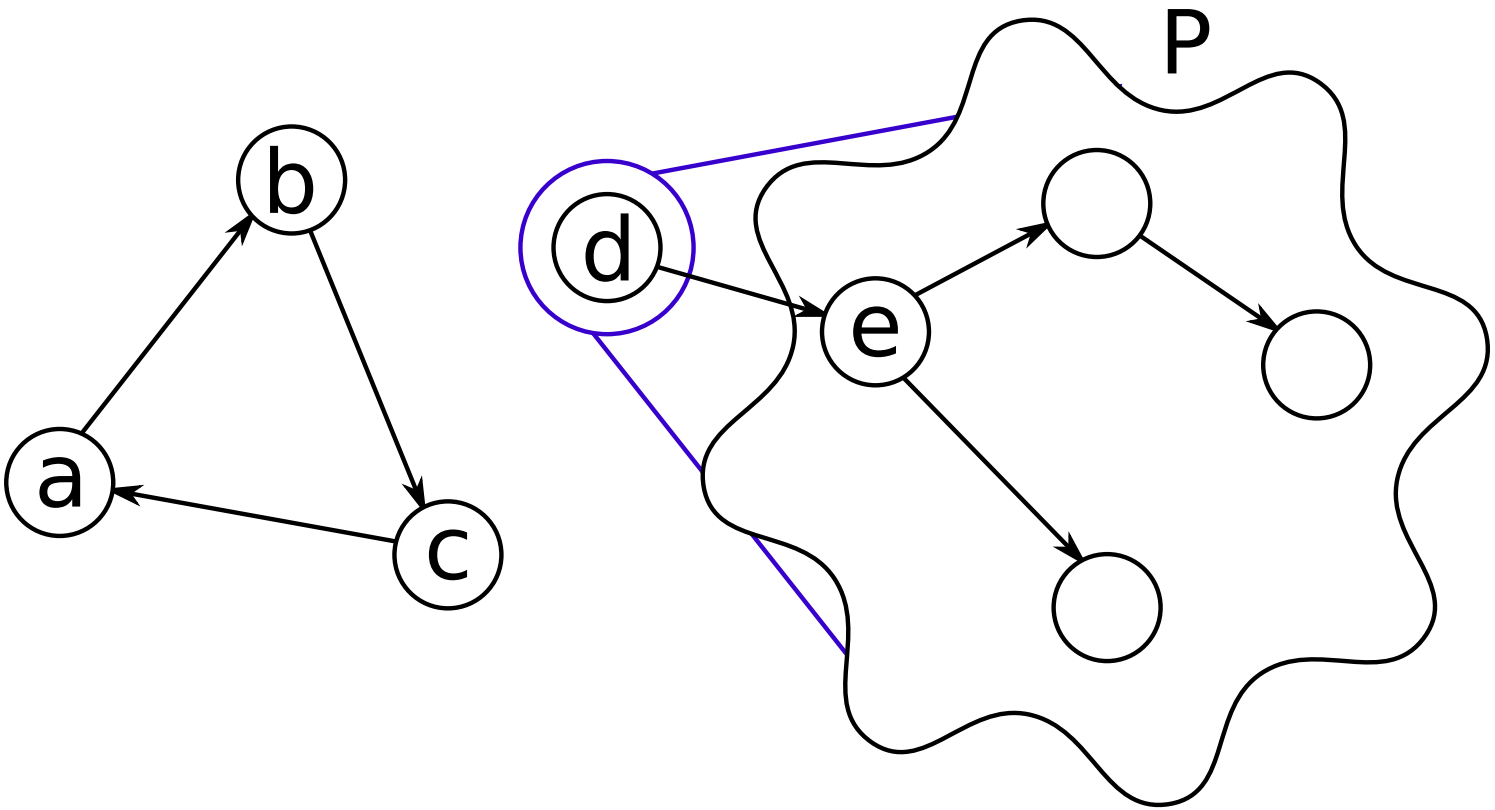
$$step[R] = done \cup next[R]$$

where $done : \mathcal{P}(claims)$, $next : \mathcal{P}(claims) \to \mathcal{P}(claims)$

$$done = \{(x, P) \mid x \in P\}$$
$$next[R] = \{(x, P) \mid \exists y.x\ S\ y \wedge (y, P) \in R\}$$

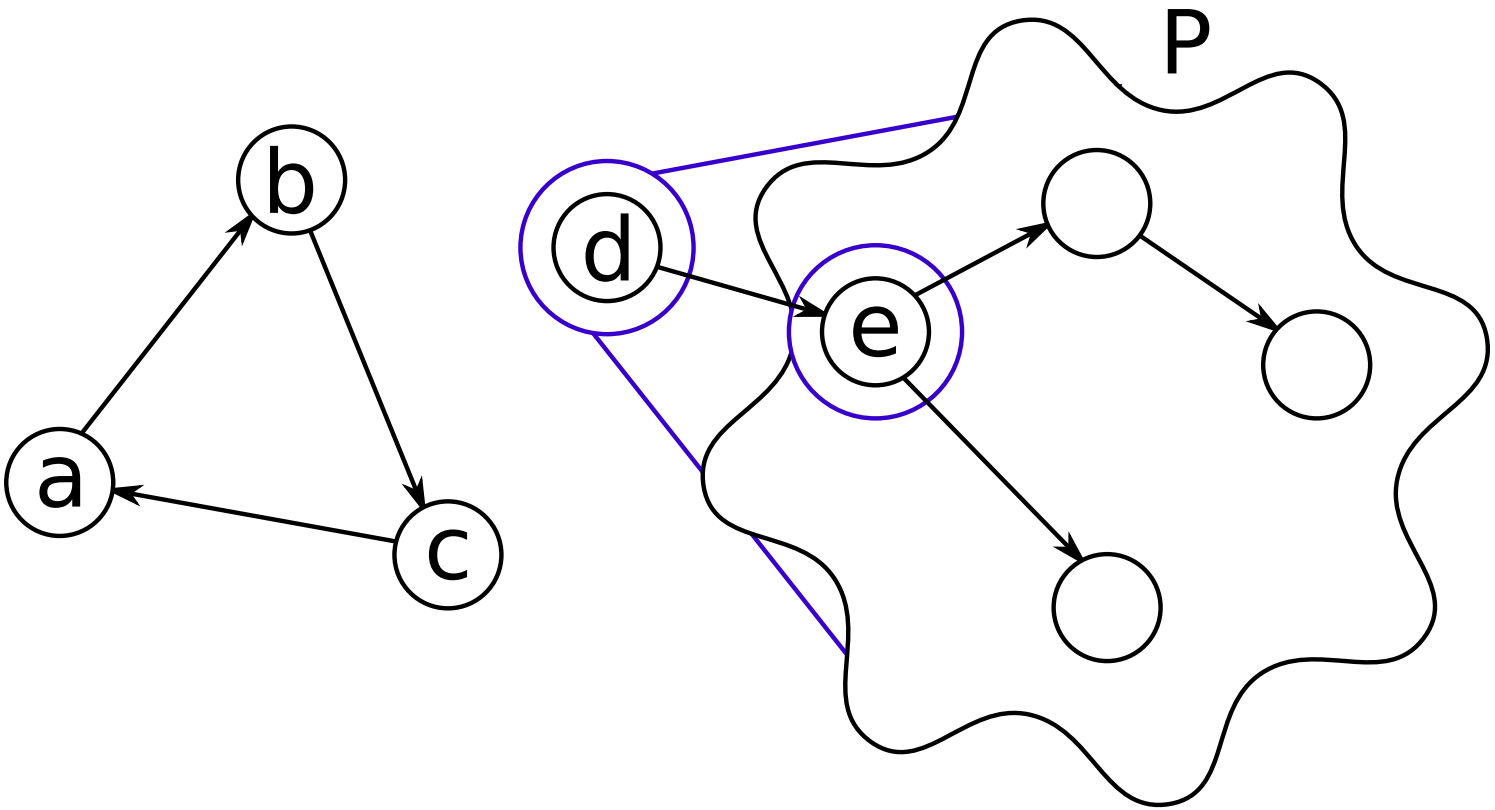
Direct coinduction 1

Goal: $d \Rightarrow P$



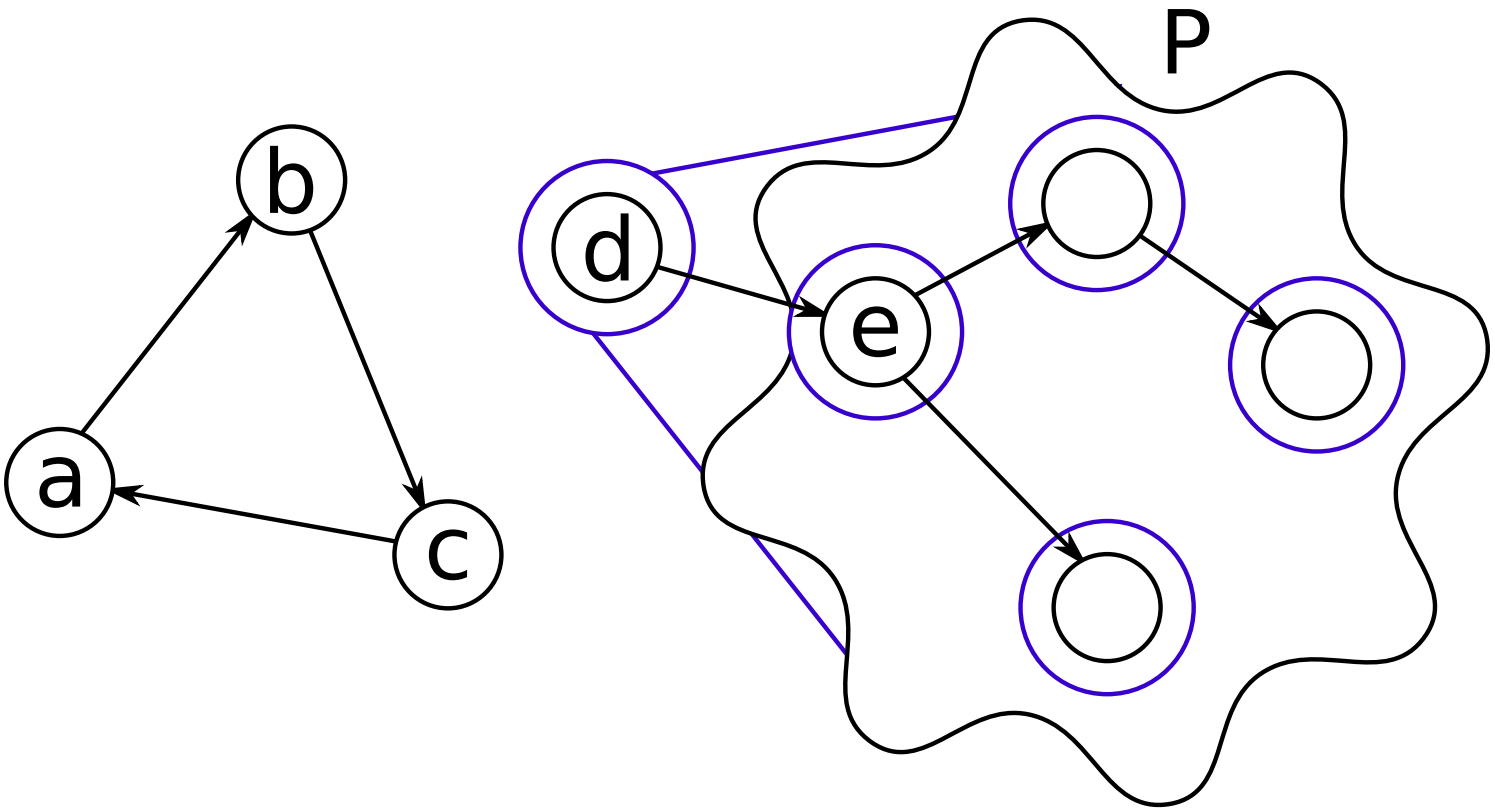
Direct coinduction 1

Expand to stable set R



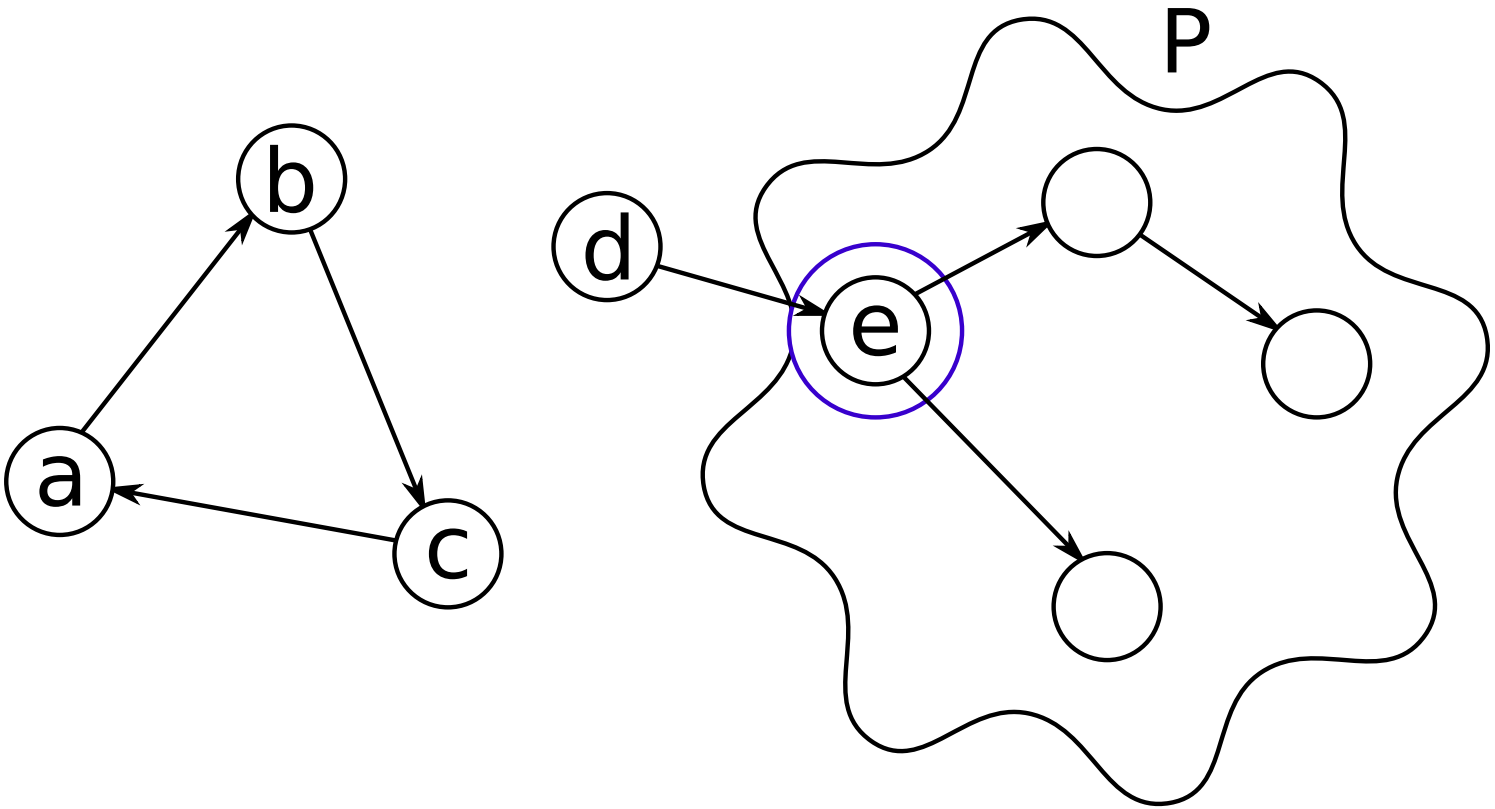
Direct coinduction 1

$step[R]$



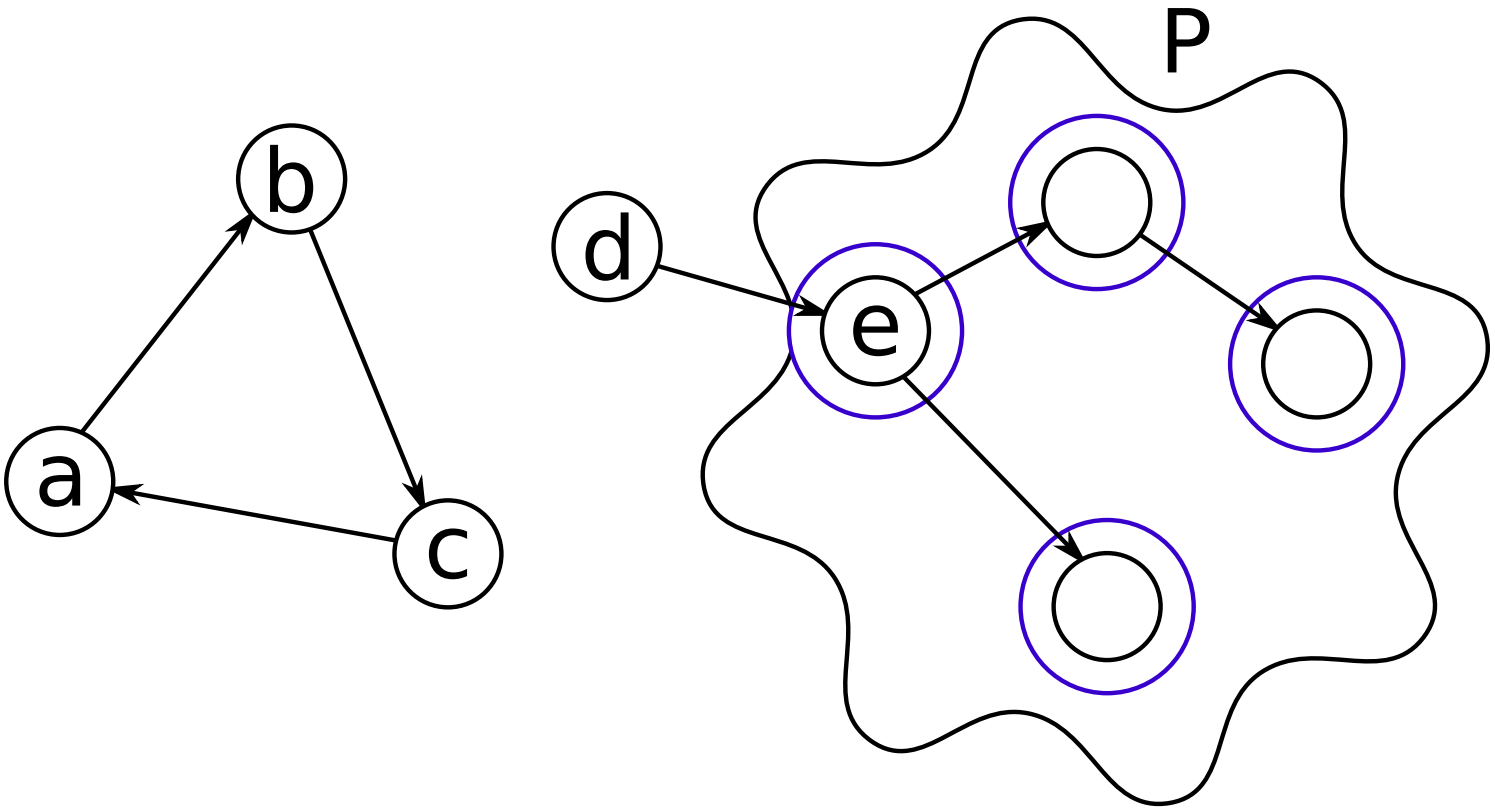
Direct coinduction 1

$(e, P) \in \mathit{done} \subseteq \mathit{step}[R]$



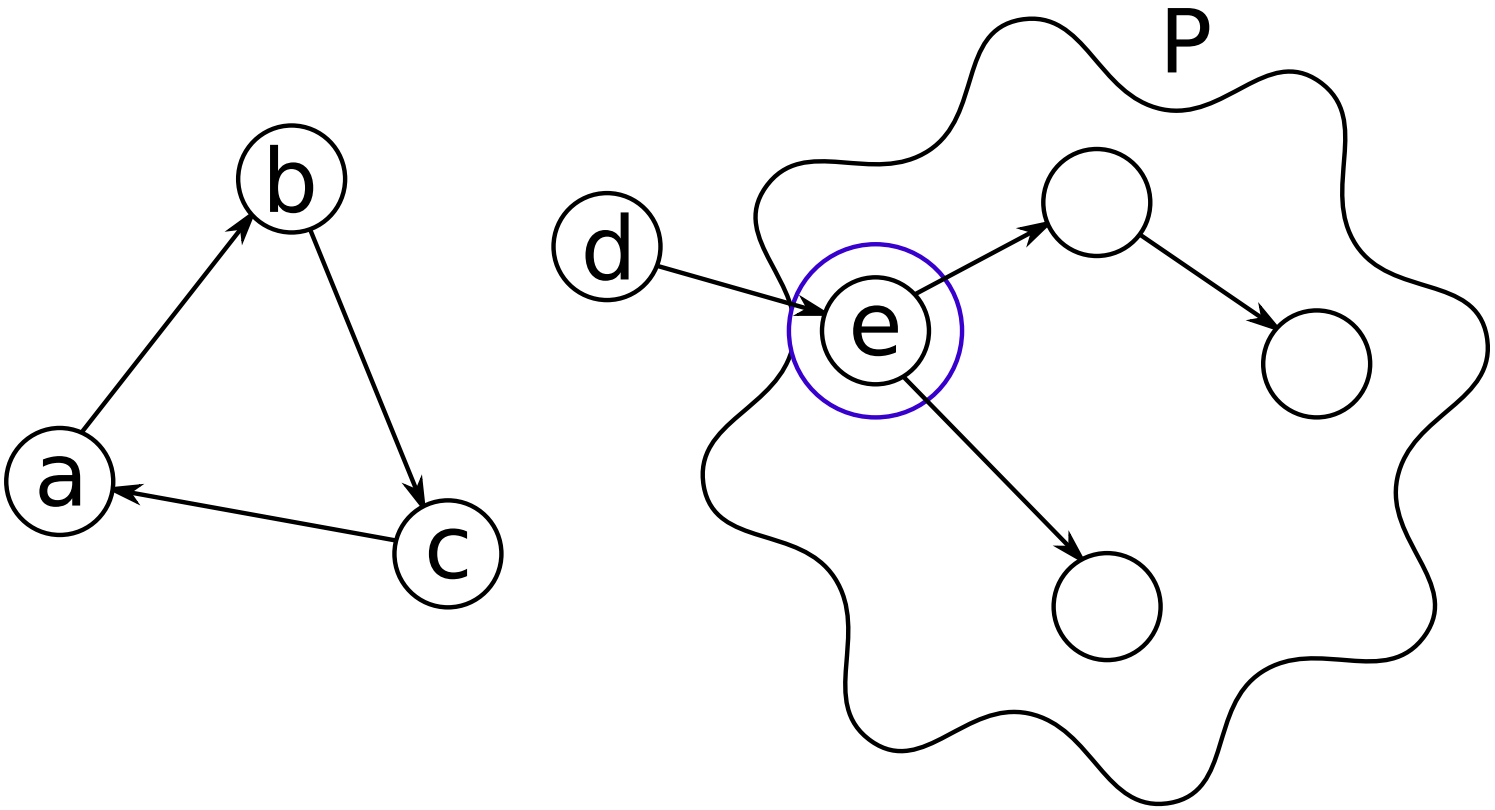
Direct coinduction 1

$(e, P) \in \mathit{done} \subseteq \mathit{step}[R]$



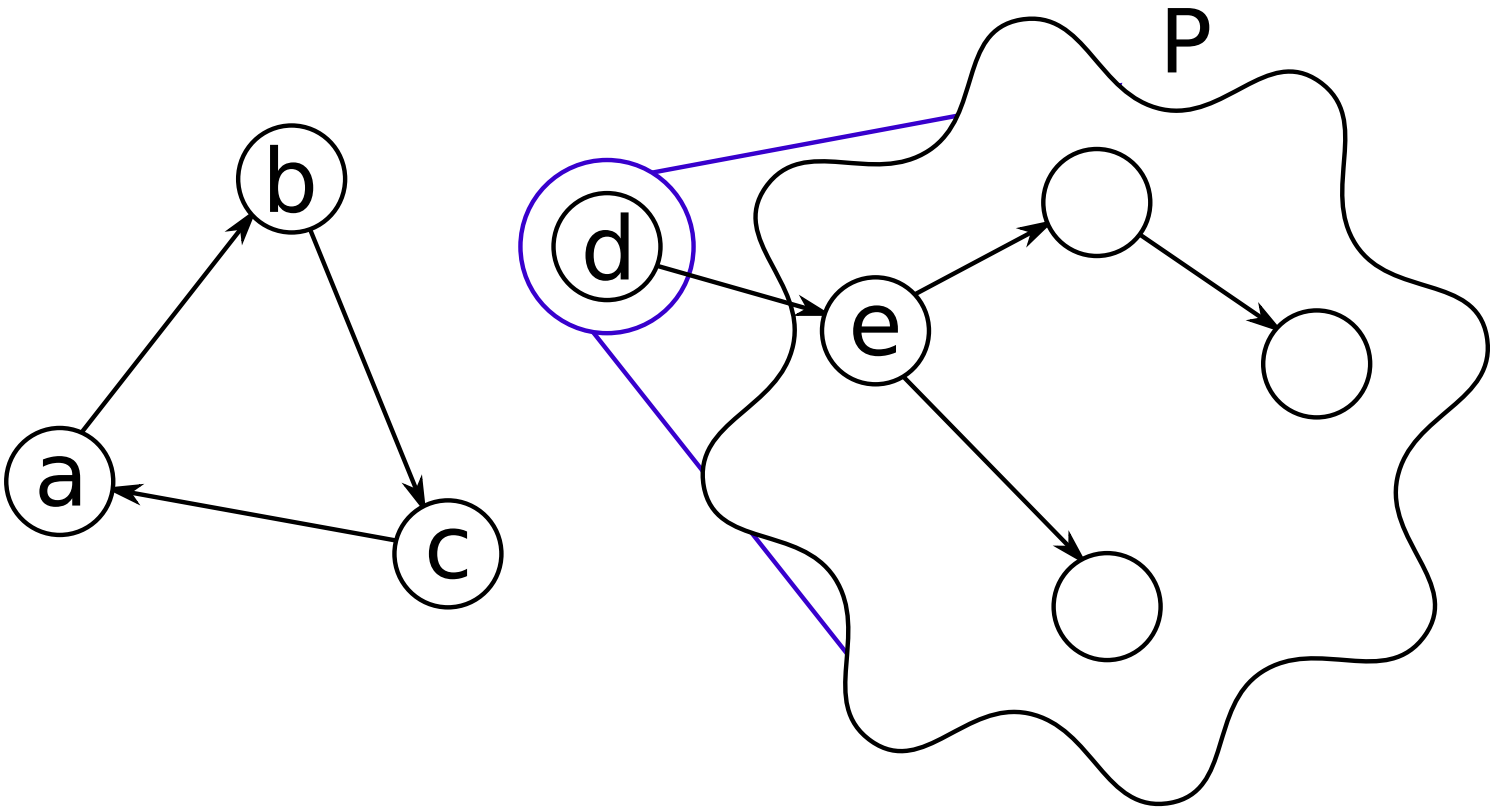
Direct coinduction 1

$(e, P) \in R \rightarrow (d, P) \in next[R] \subseteq step[R]$



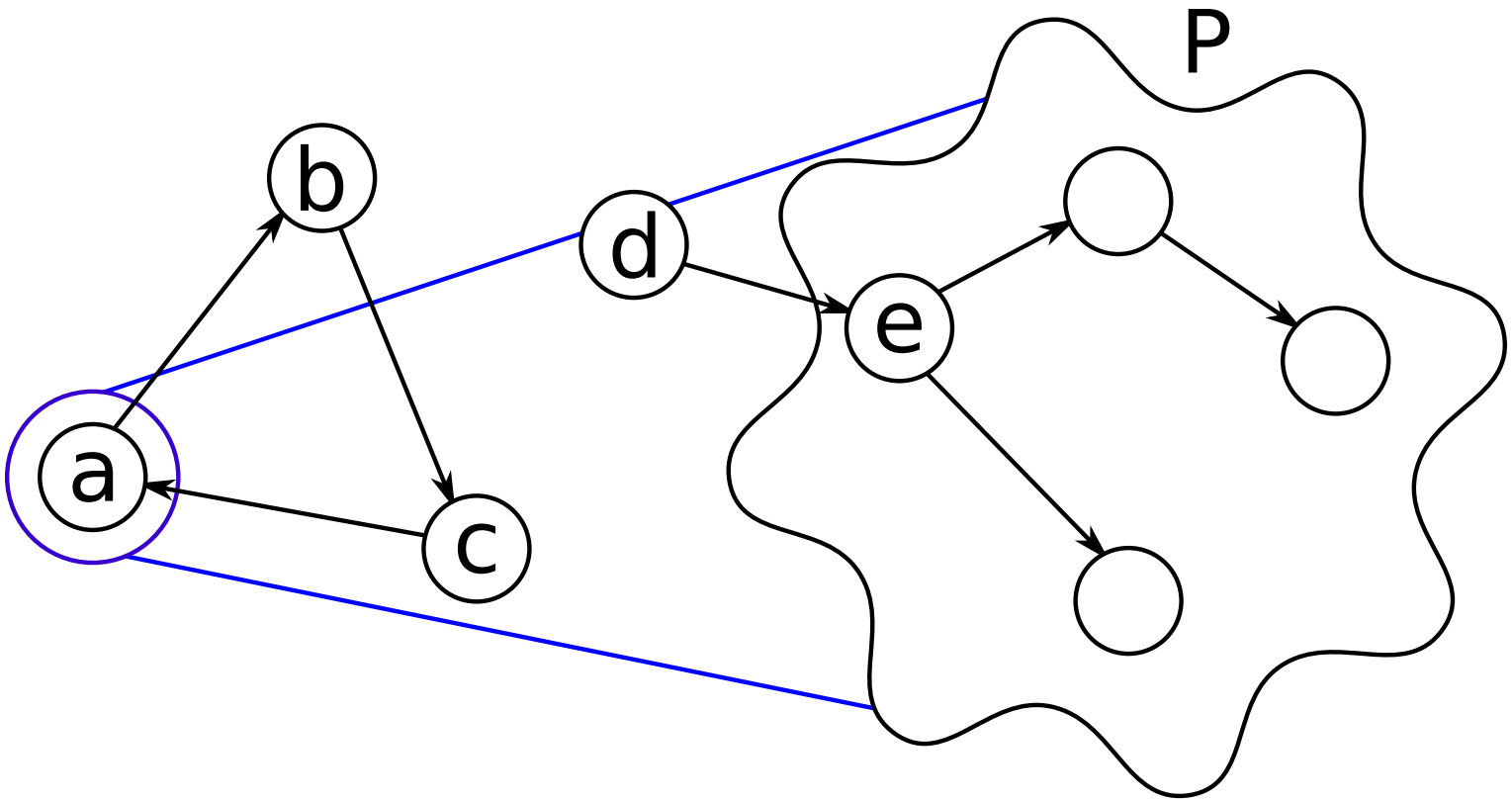
Direct coinduction 1

$(e, P) \in R \rightarrow (d, P) \in next[R] \subseteq step[R]$



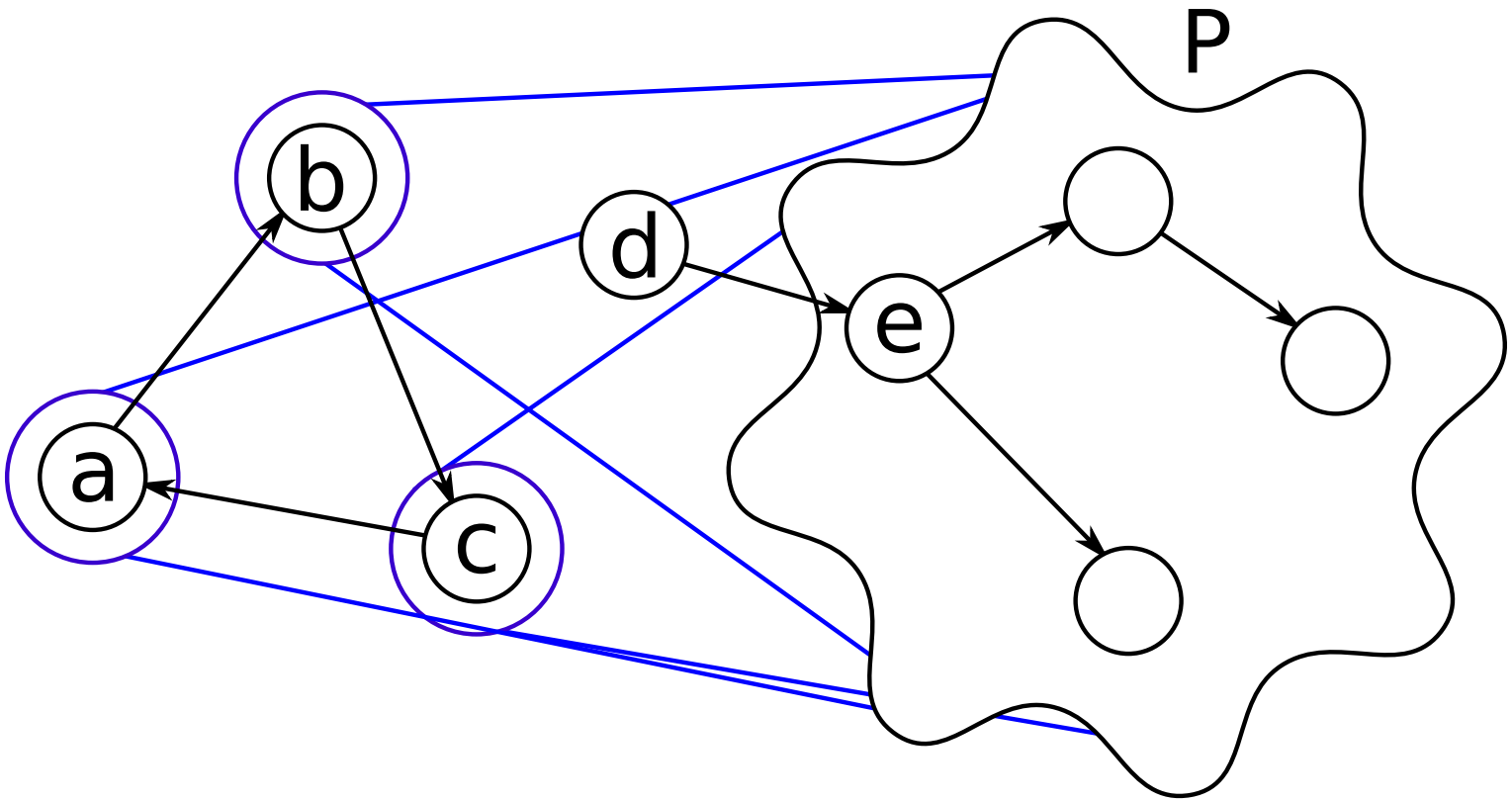
Direct coinduction 2

Goal: $a \Rightarrow P$



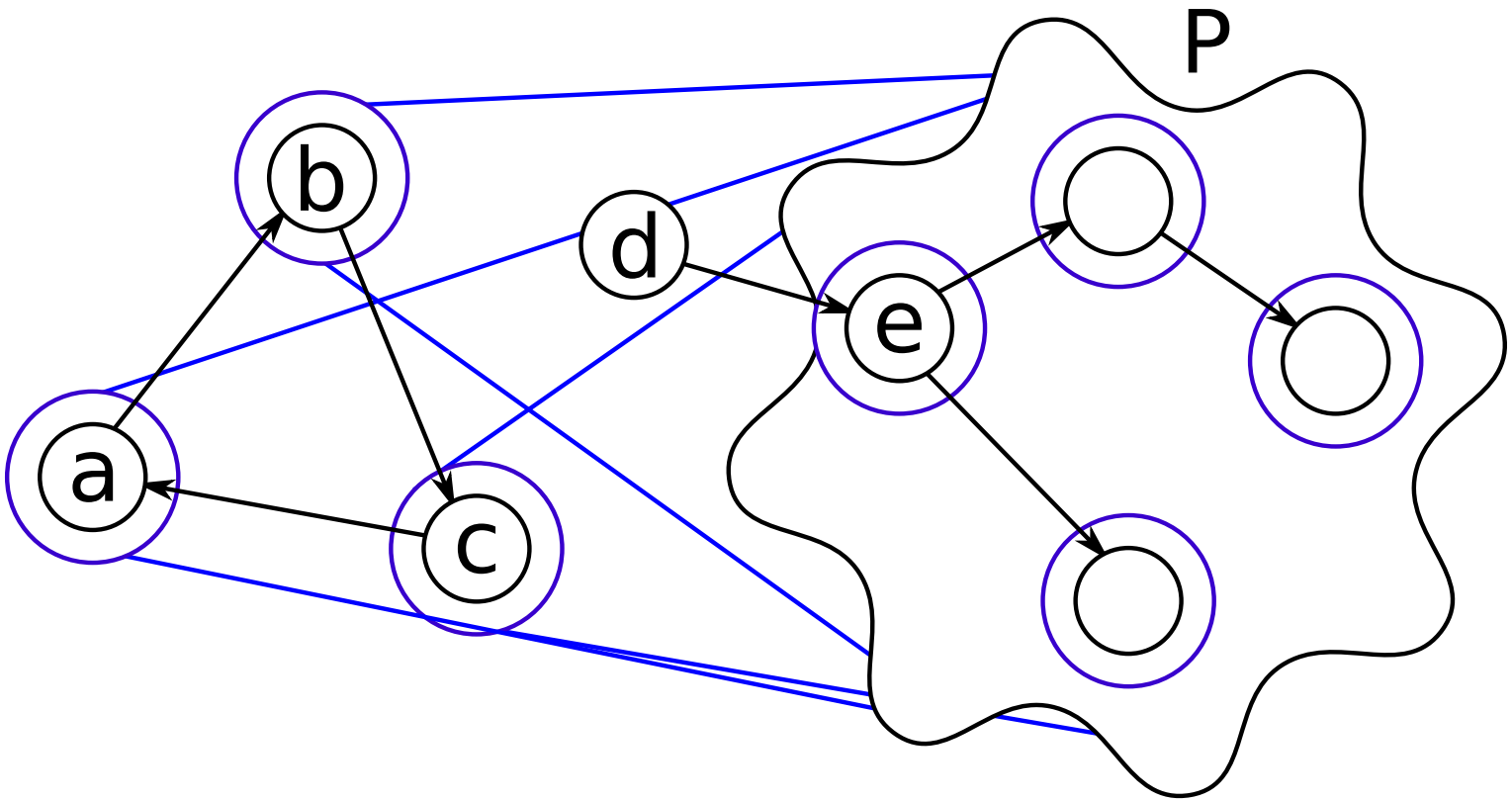
Direct coinduction 2

Expand to stable set R



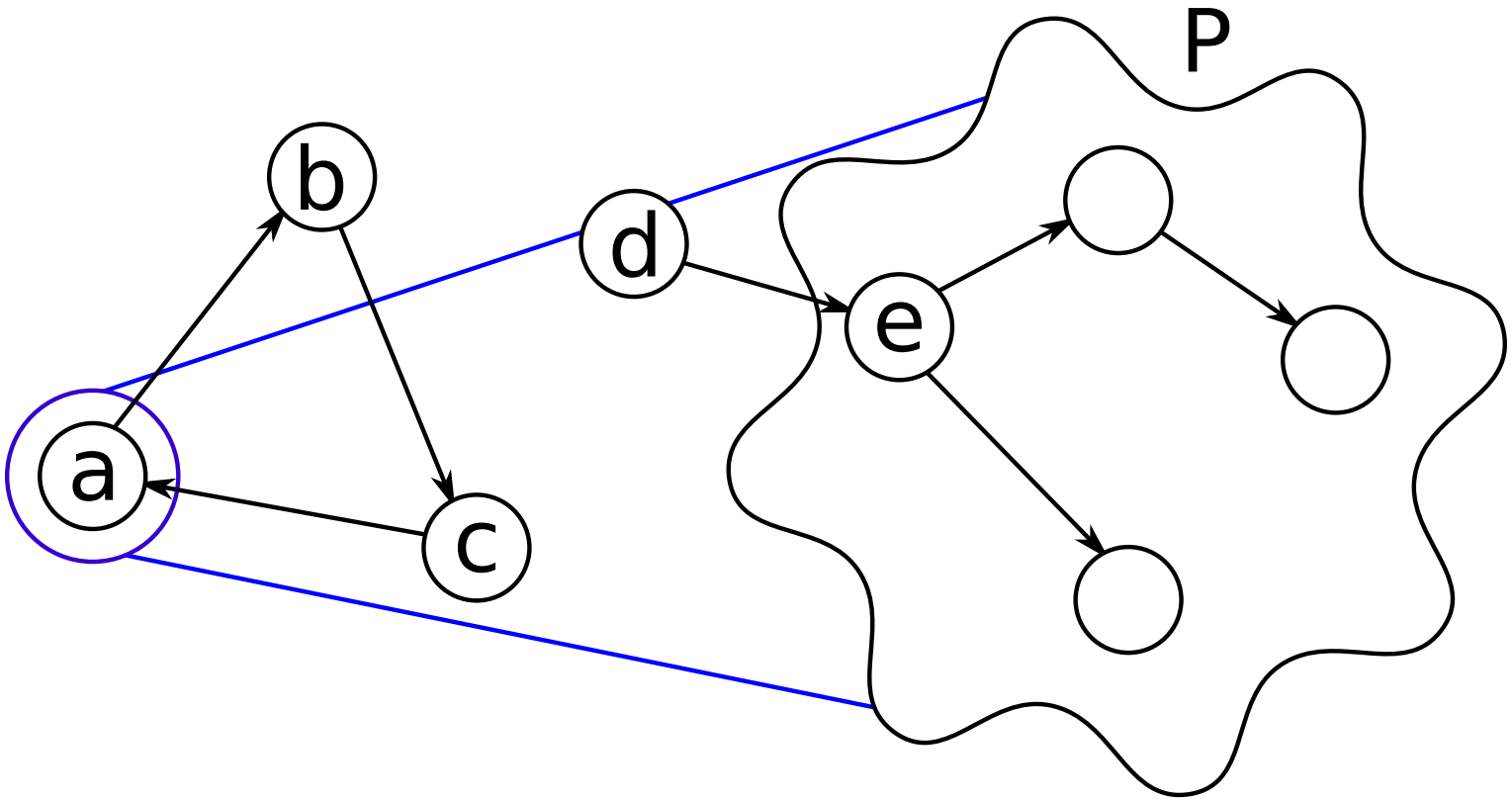
Direct coinduction 2

$step[R]$



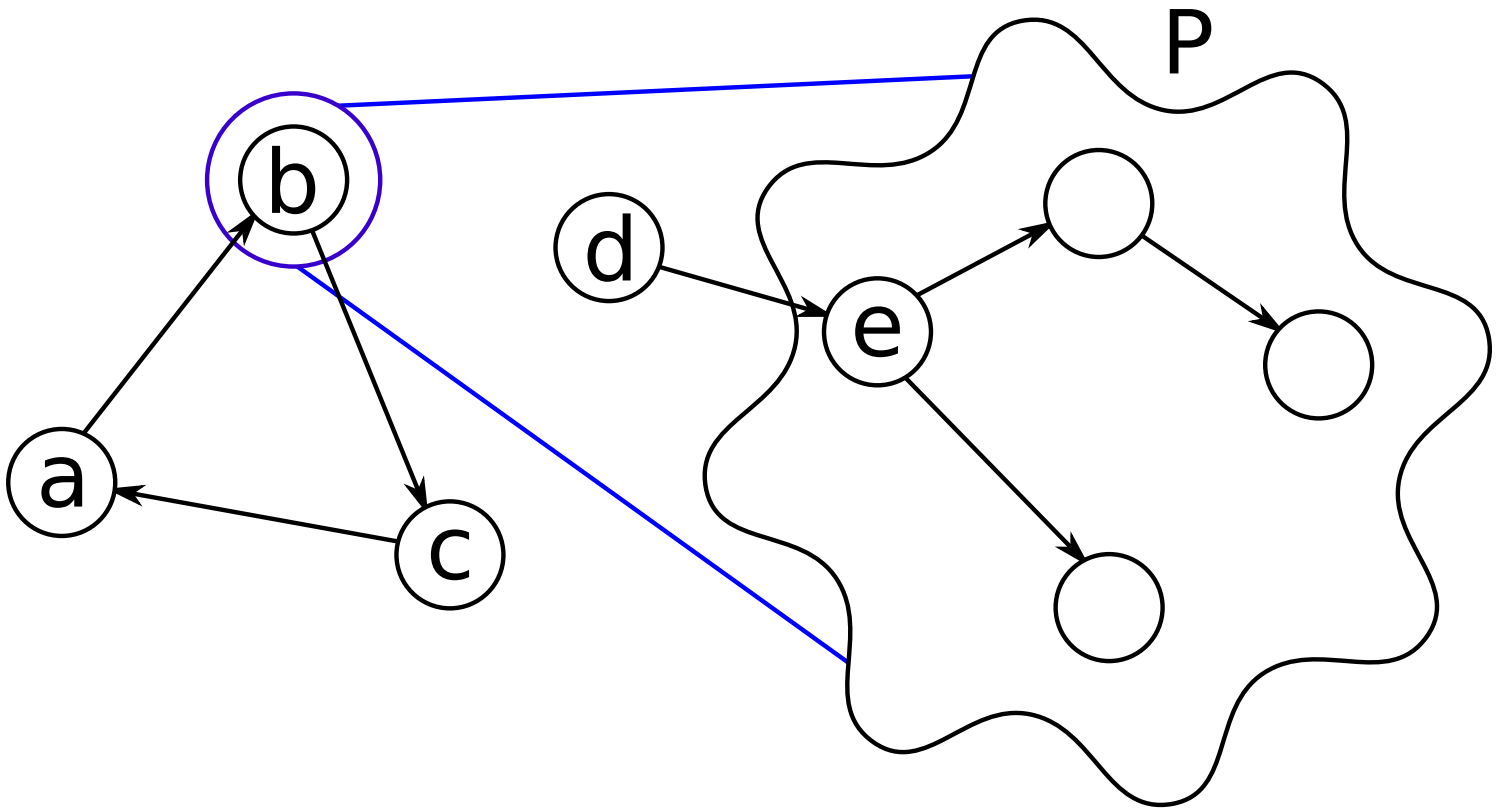
Direct coinduction 2

$(b, P) \in R \rightarrow (a, P) \in next[R] \subseteq step[R]$



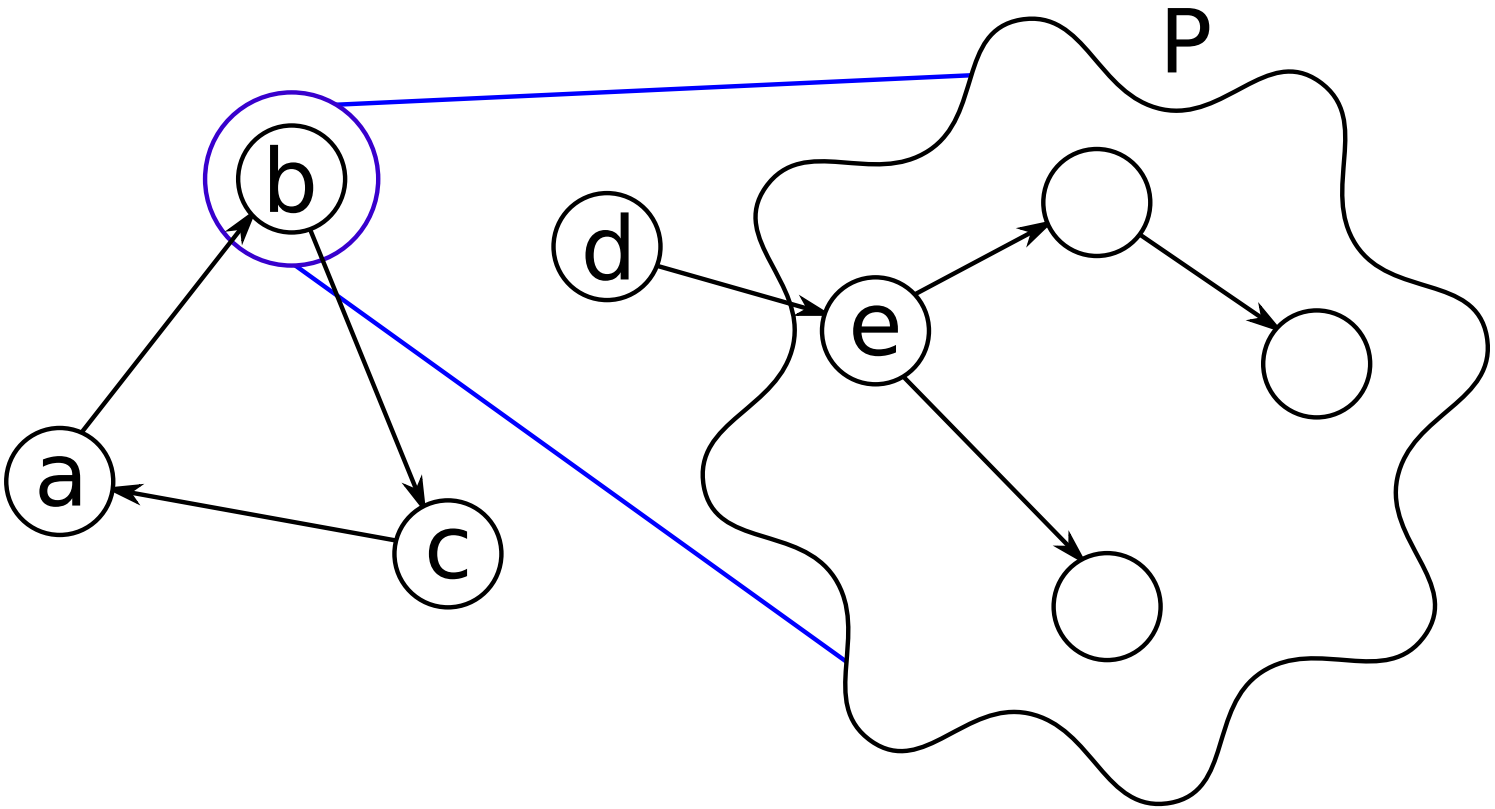
Direct coinduction 2

$(b, P) \in R \rightarrow (a, P) \in \mathit{next}[R] \subseteq \mathit{step}[R]$



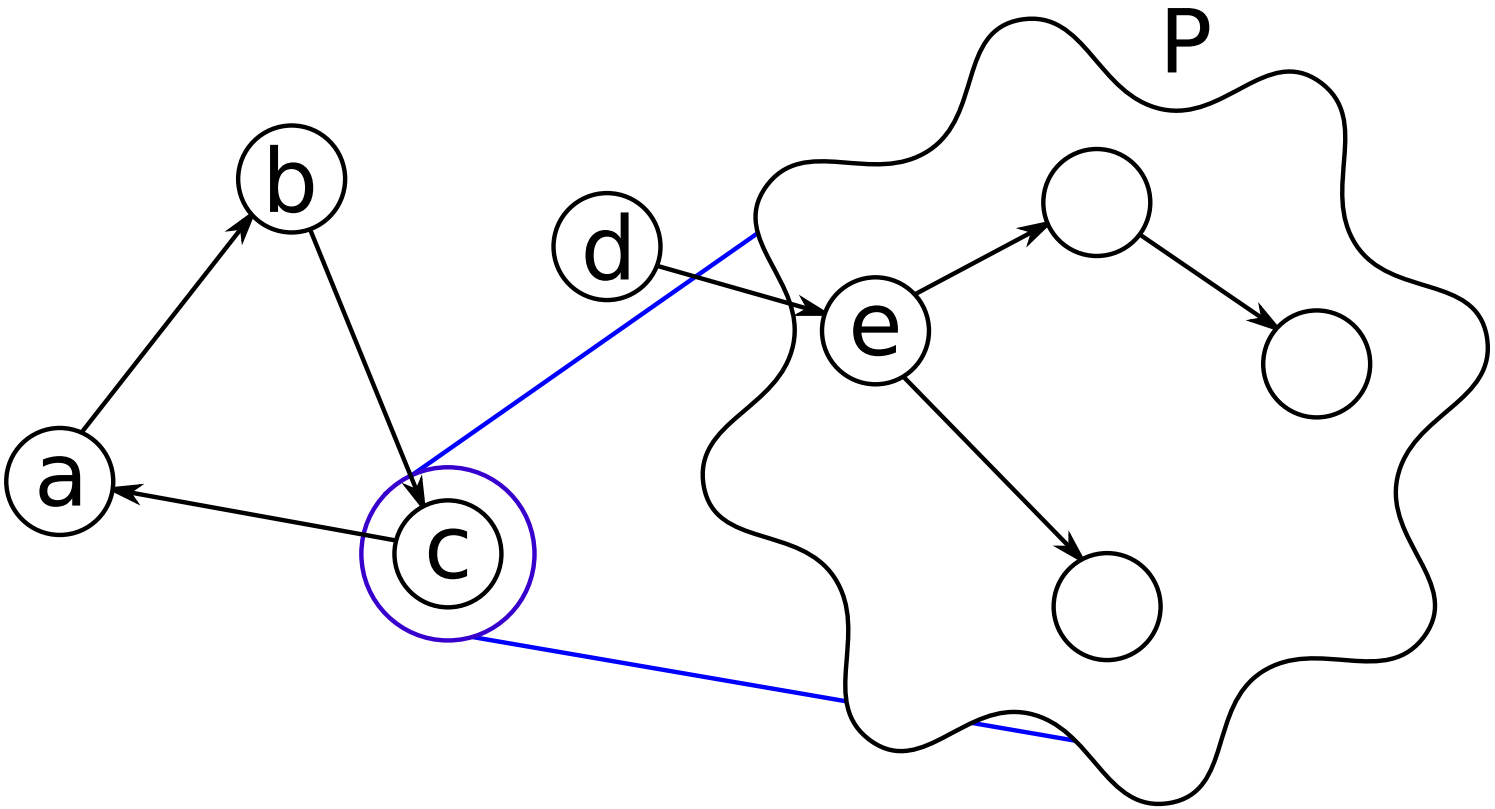
Direct coinduction 2

$(c, P) \in R \rightarrow (b, P) \in next[R] \subseteq step[R]$



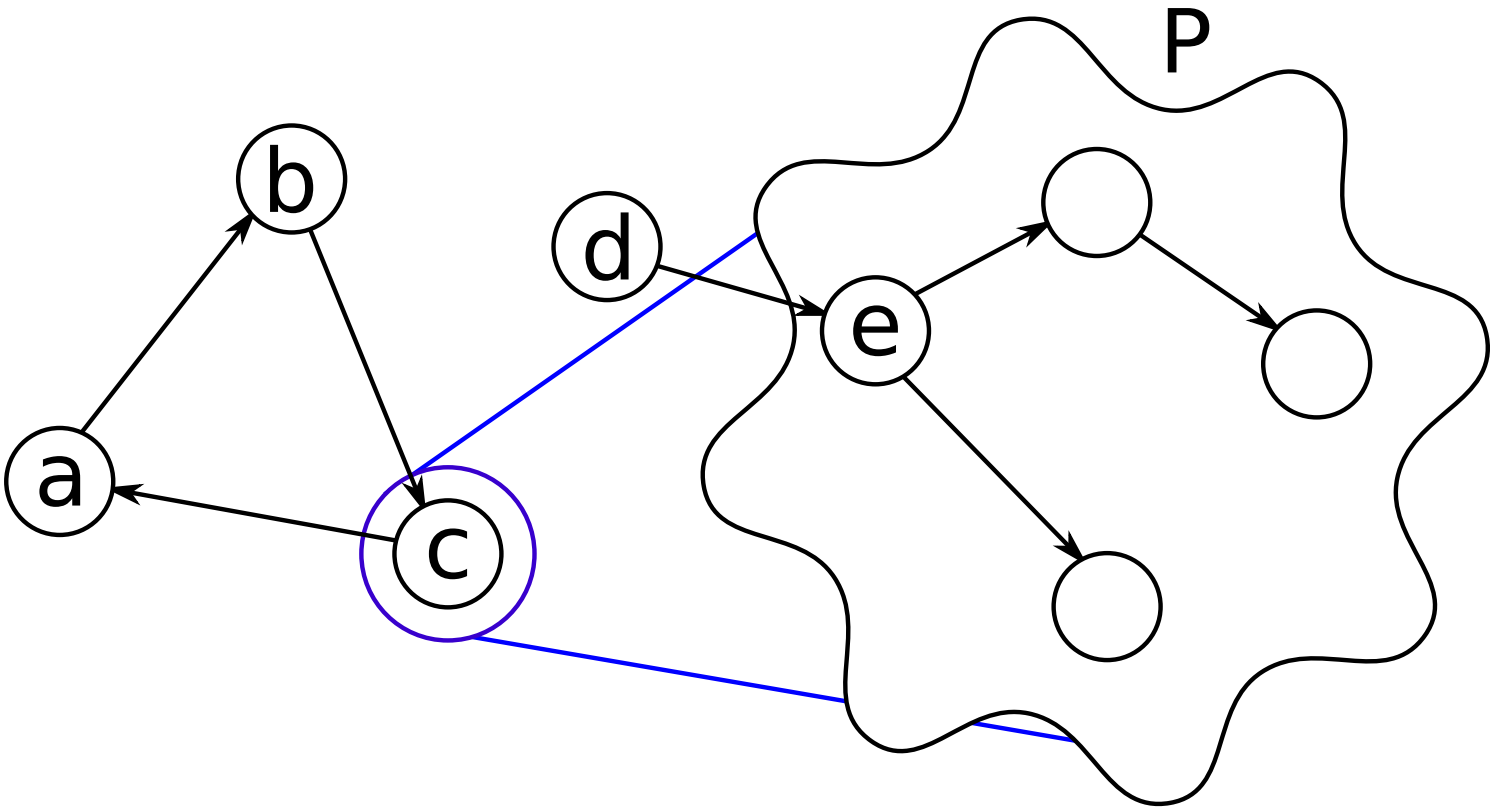
Direct coinduction 2

$(c, P) \in R \rightarrow (b, P) \in next[R] \subseteq step[R]$



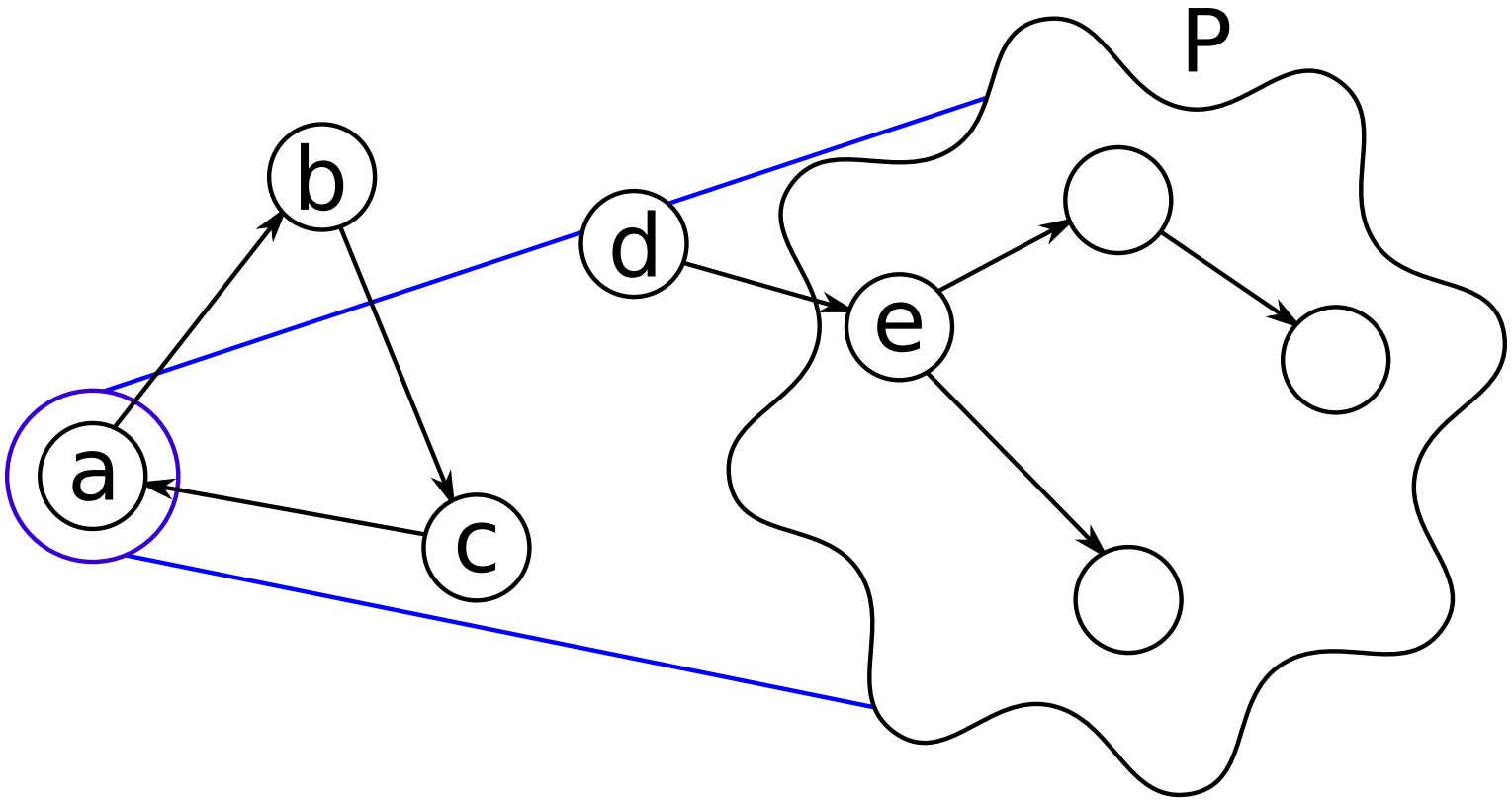
Direct coinduction 2

$(a, P) \in R \rightarrow (c, P) \in next[R] \subseteq step[R]$



Direct coinduction 2

$(a, P) \in R \rightarrow (c, P) \in next[R] \subseteq step[R]$



Direction coinduction on a program

To prove

$$\langle \texttt{while(n!=0) \{n=n-1\}}, \{\texttt{n} \mapsto n_0\}\rangle \Rightarrow \{\langle \texttt{skip}, \{\texttt{n} \mapsto 0\}\rangle\}$$

must also claim

$$\langle \texttt{n=n-1; while(n!=0) \{n=n-1\}}, \{\texttt{n} \mapsto n_0\}\rangle \Rightarrow \{\langle \texttt{skip}, \{\texttt{n} \mapsto 0\}\rangle\}$$

and

$$\langle \texttt{skip}, \{\texttt{n} \mapsto 0\}\rangle \Rightarrow \{\langle \texttt{skip}, \{\texttt{n} \mapsto 0\}\rangle\}$$

Recovering Rules

- Direct coinduction tedious
- Replacements for proof rules
  - Transitivity, Weakening, Assertion, etc.
- Combine freely

Multi-step coinduction

Given $R : \mathcal{P}(claims)$, close under *step* by fixpoint (construction)

$$\mu C.R \cup step[C]$$

Definition

$R : \mathcal{P}(claims)$ is *step-stable using multiple steps* if

$$R \subseteq step[\mu C.R \cup step[C]]$$

Multi-step coinduction

If $R : \mathcal{P}(claims)$ has $R \subseteq step[\mu C.R \cup step[C]]$ then

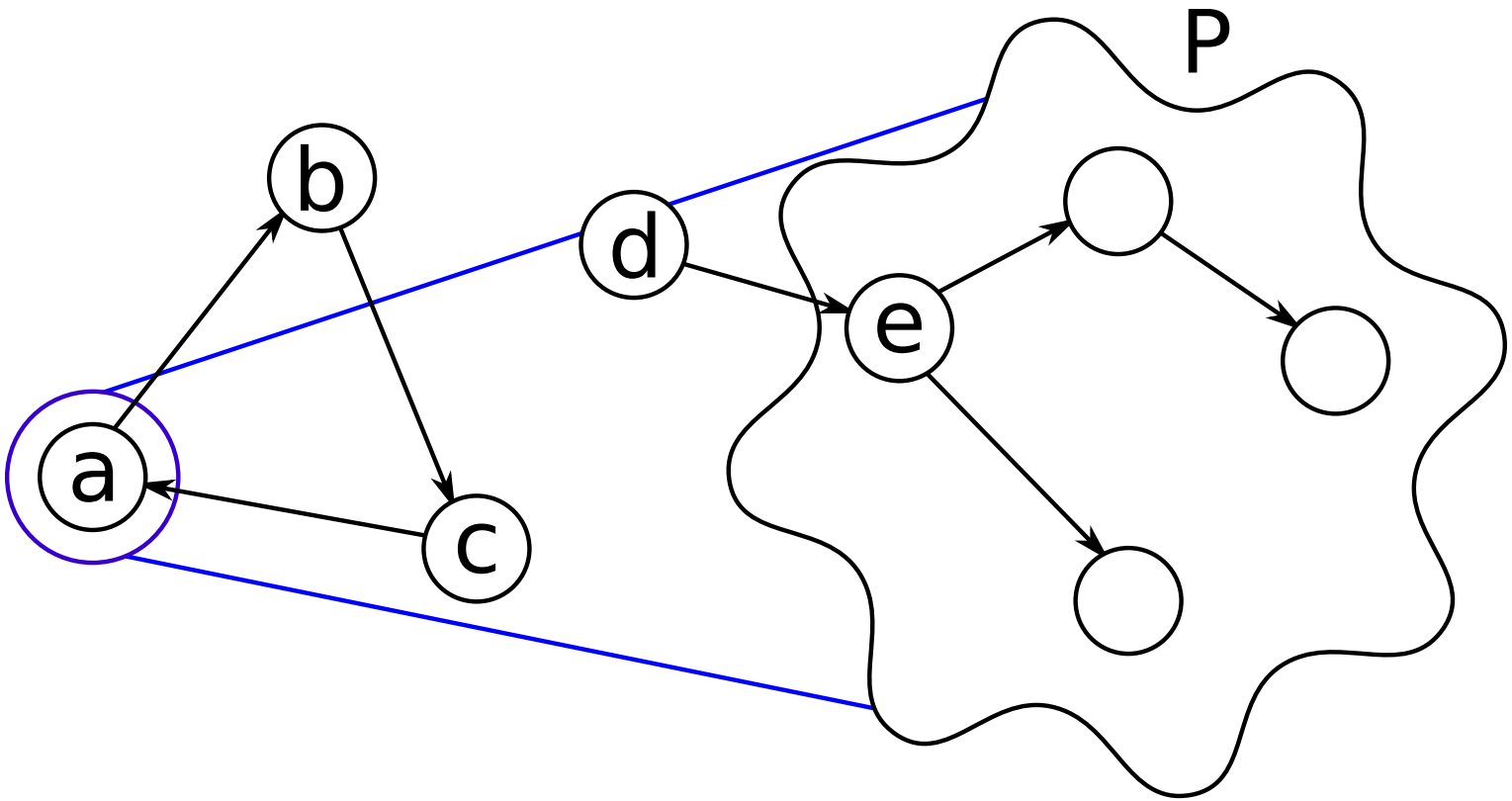
$$(\mu C.R \cup step[C]) \subseteq step[\mu C.R \cup step[C]]$$

and thus

$$R \subseteq (\mu C.R \cup step[C]) \subseteq reaches$$

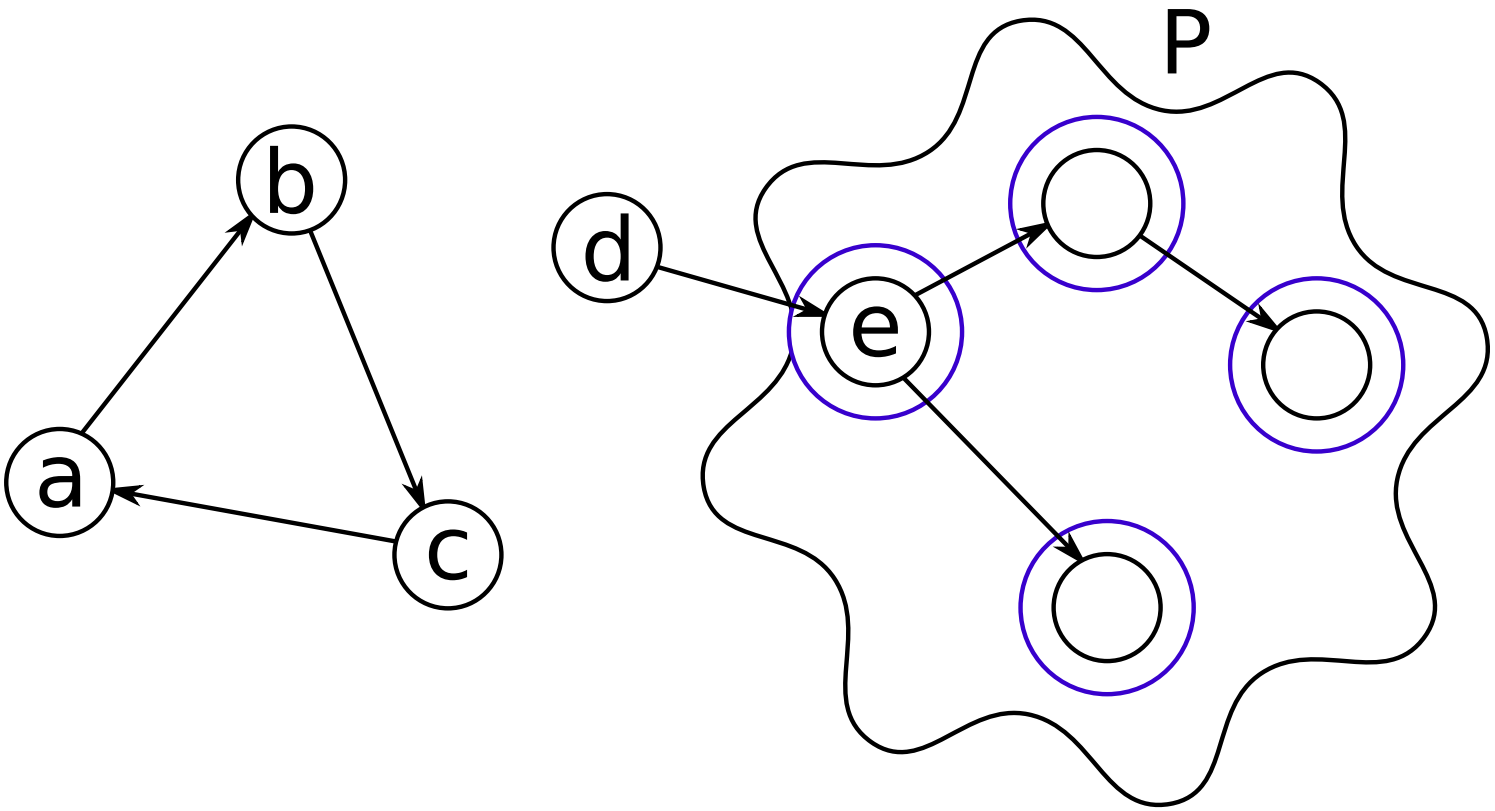
Proving with multiple steps

Goal: $R = \{(a, P)\}$



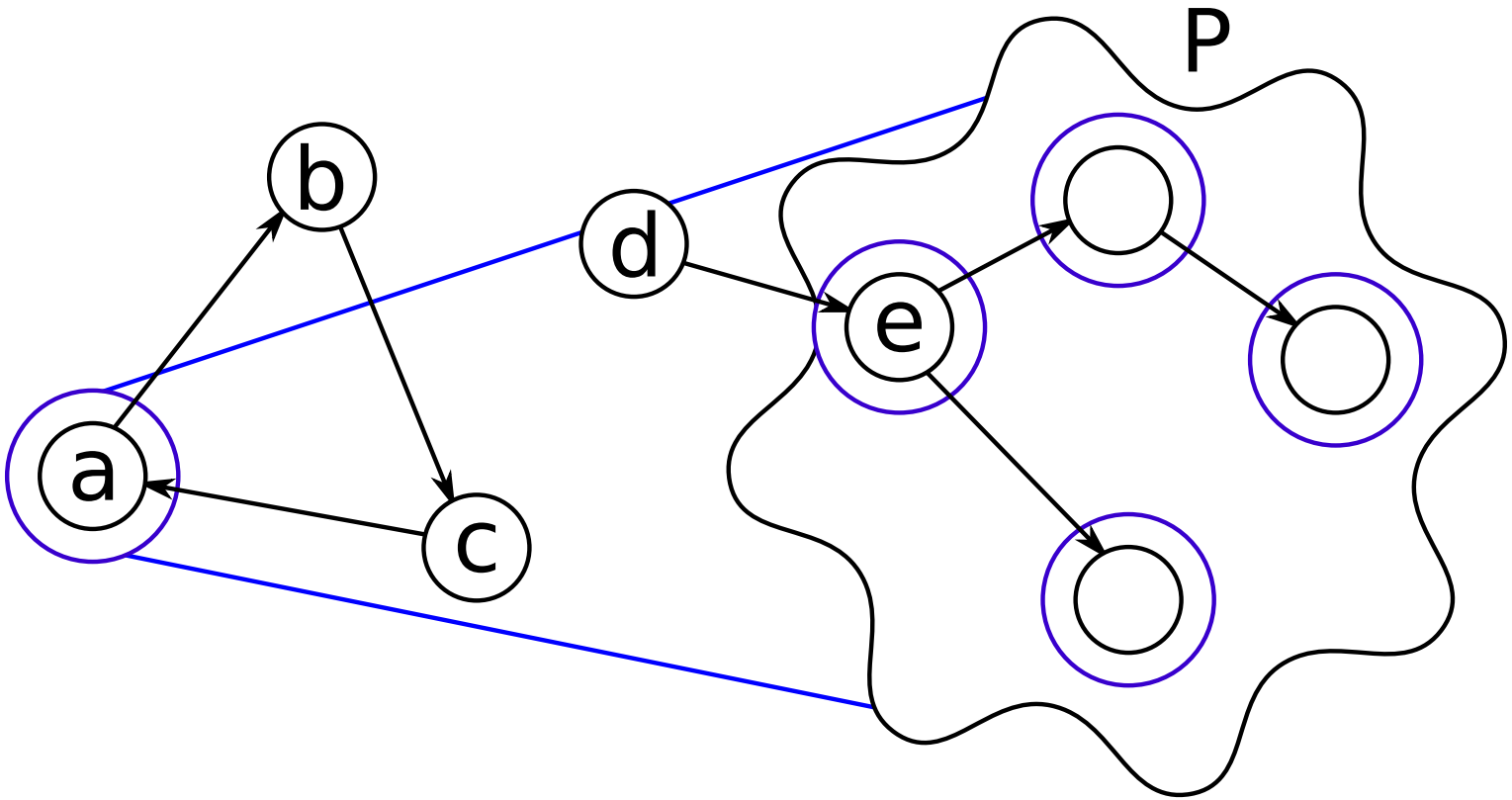
Proving with multiple steps

Closing: $step[\emptyset]$



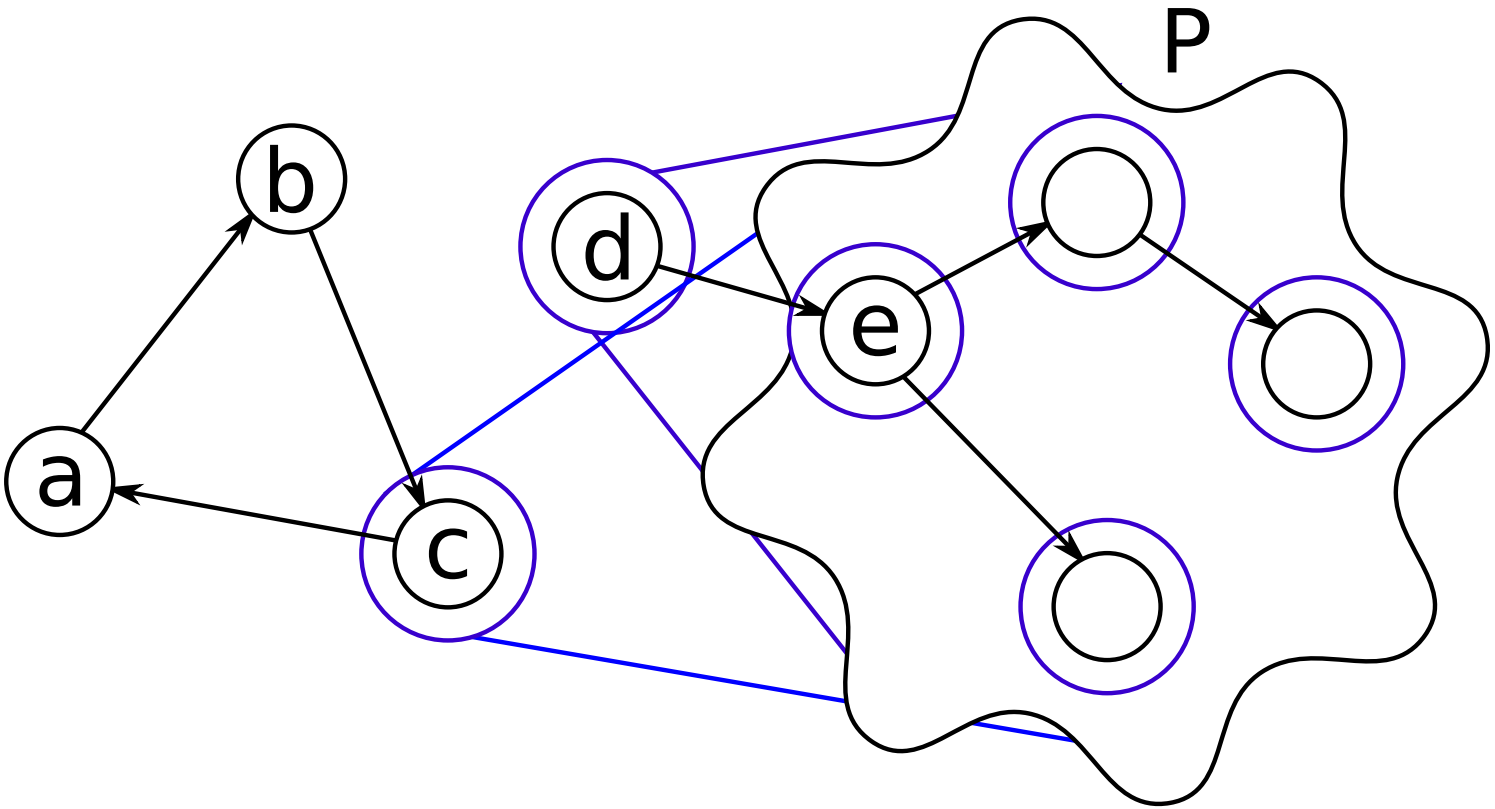
Proving with multiple steps

Closing: $R \cup step[\emptyset]$



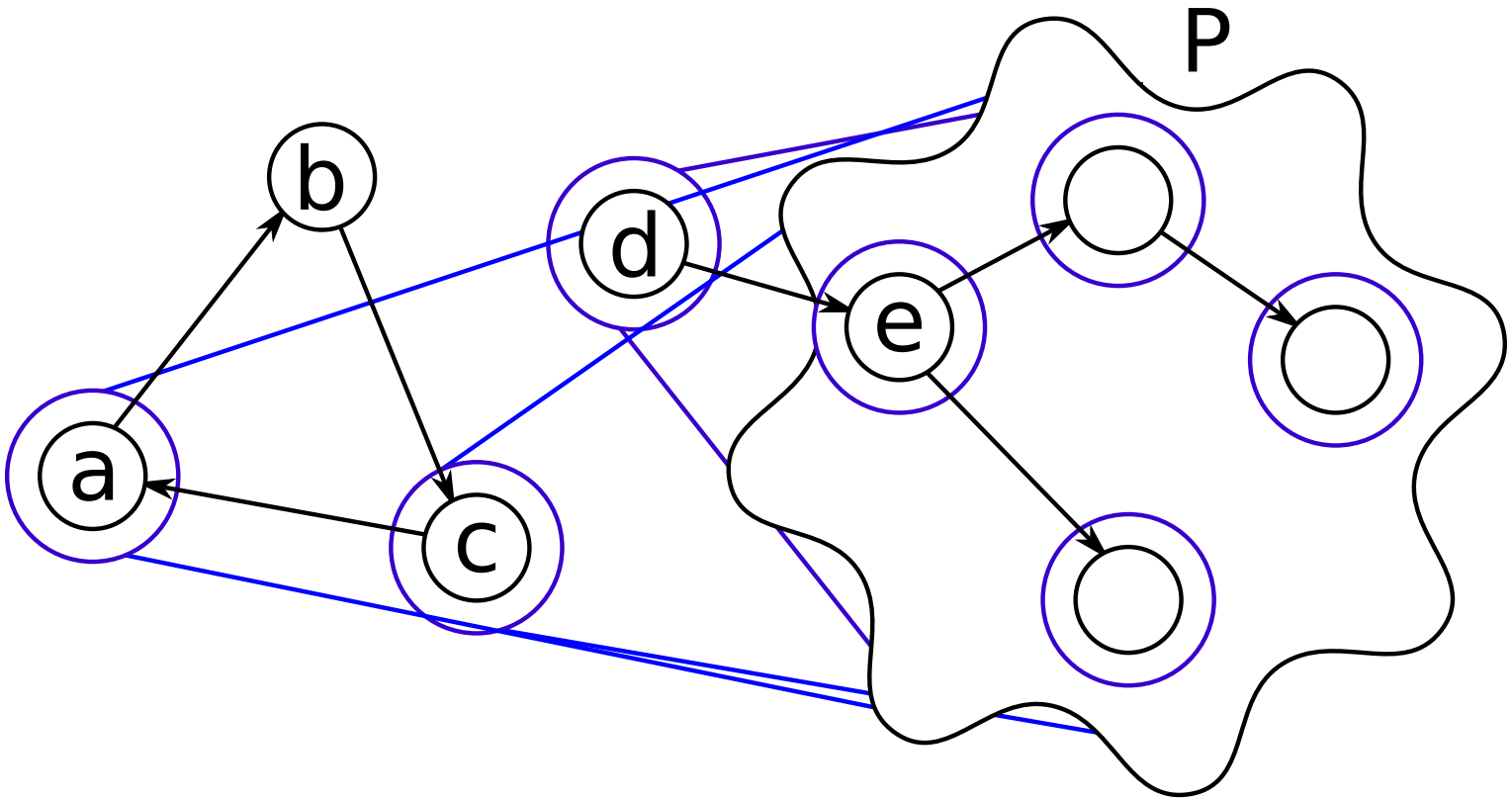
Proving with multiple steps

Closing: $step[R \cup step[\emptyset]]$



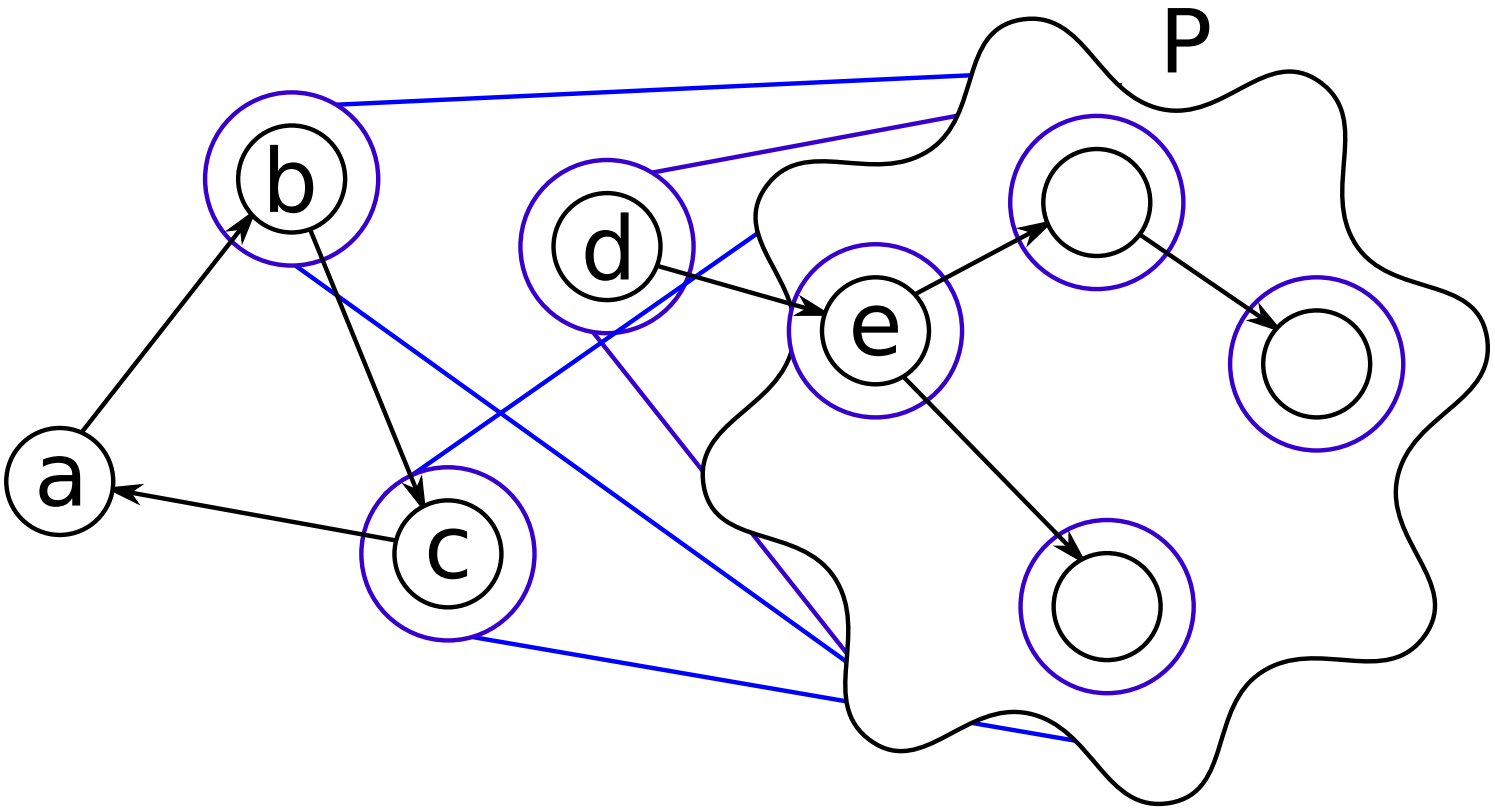
Proving with multiple steps

Closing: $R \cup step[R \cup step[\emptyset]]$



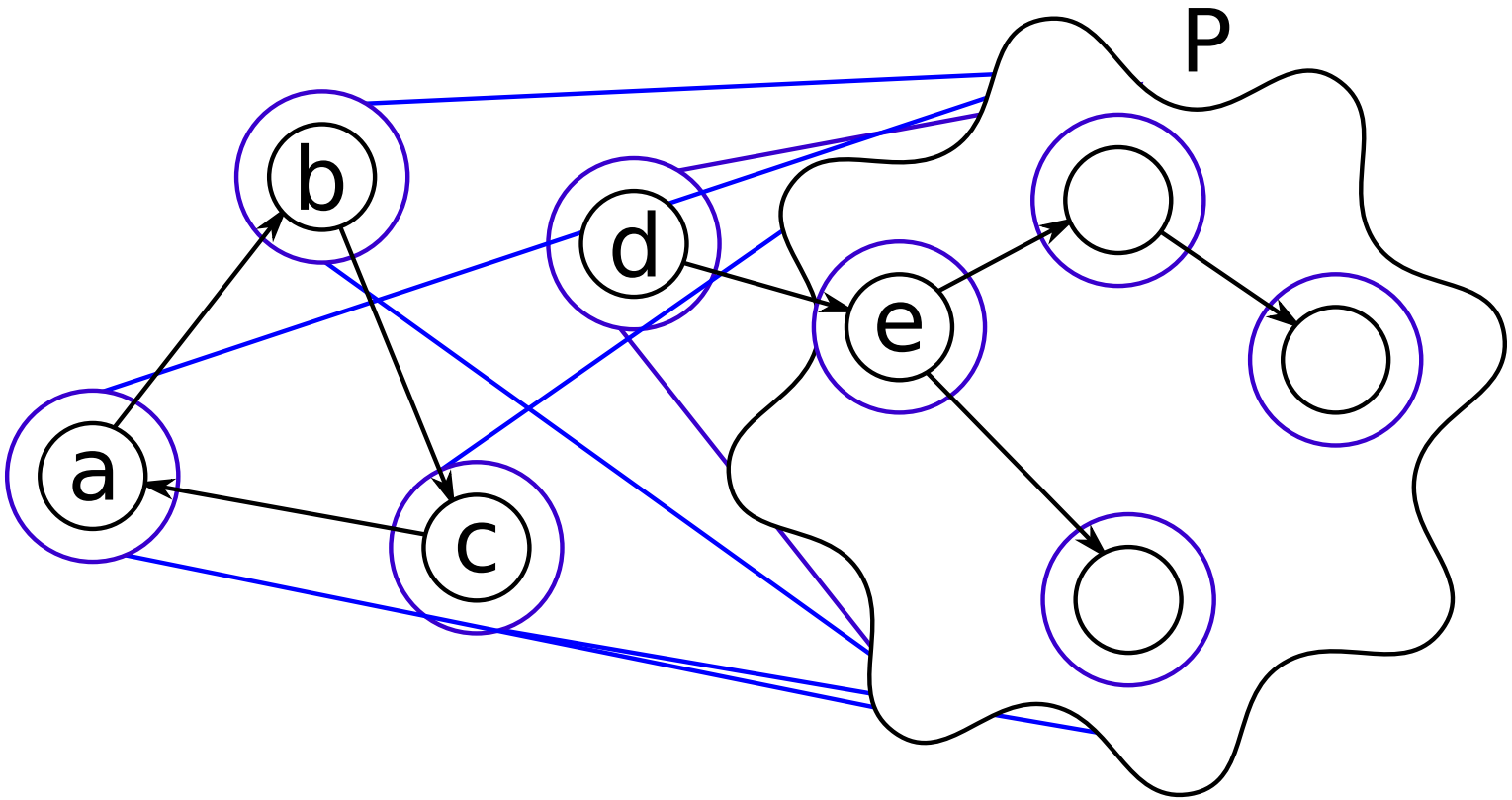
Proving with multiple steps

Closing: $step[R \cup step[R \cup step[\emptyset]]]$



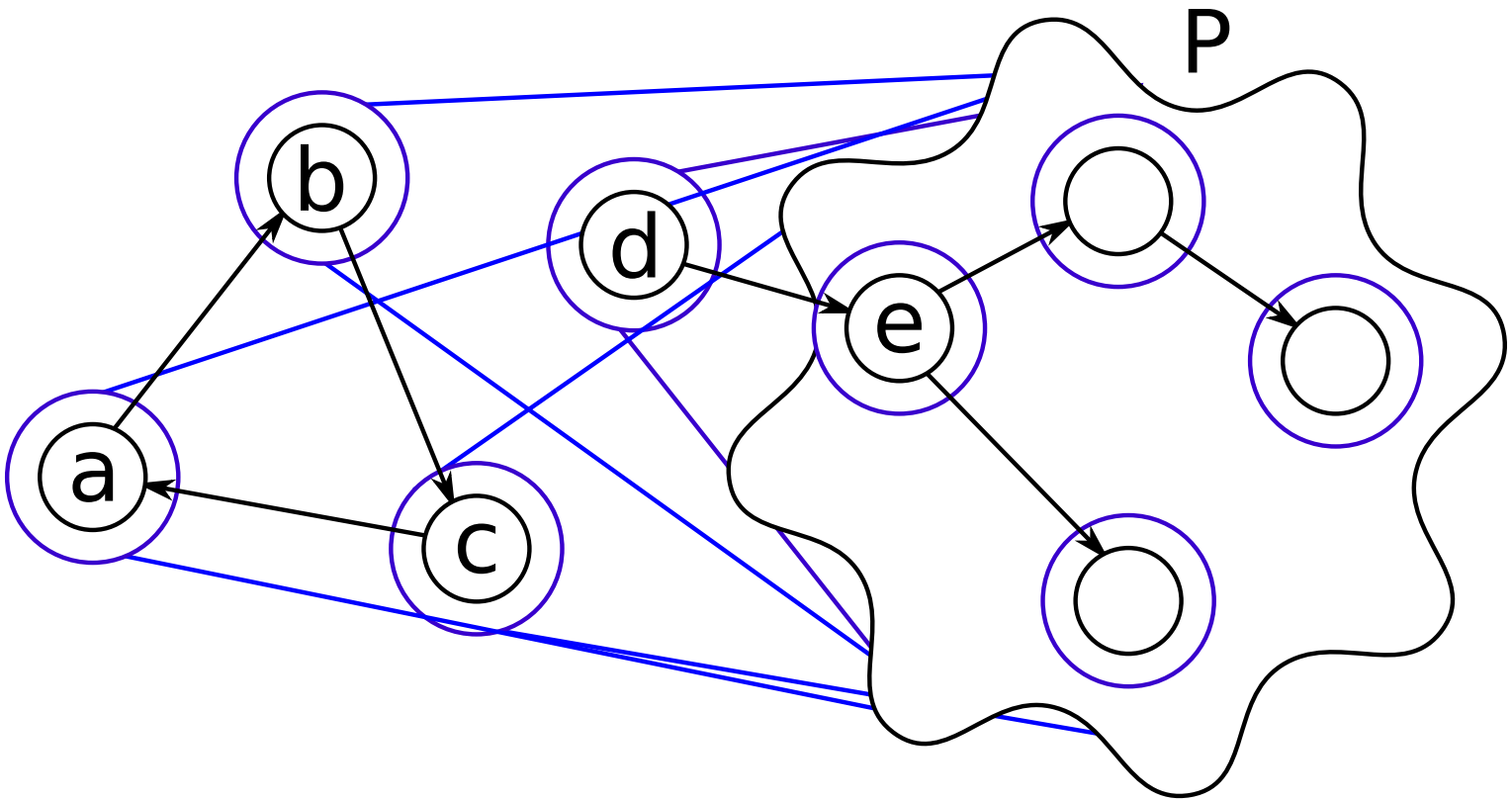
Proving with multiple steps

Closing: $R \cup step[R \cup step[R \cup step[\emptyset]]]$



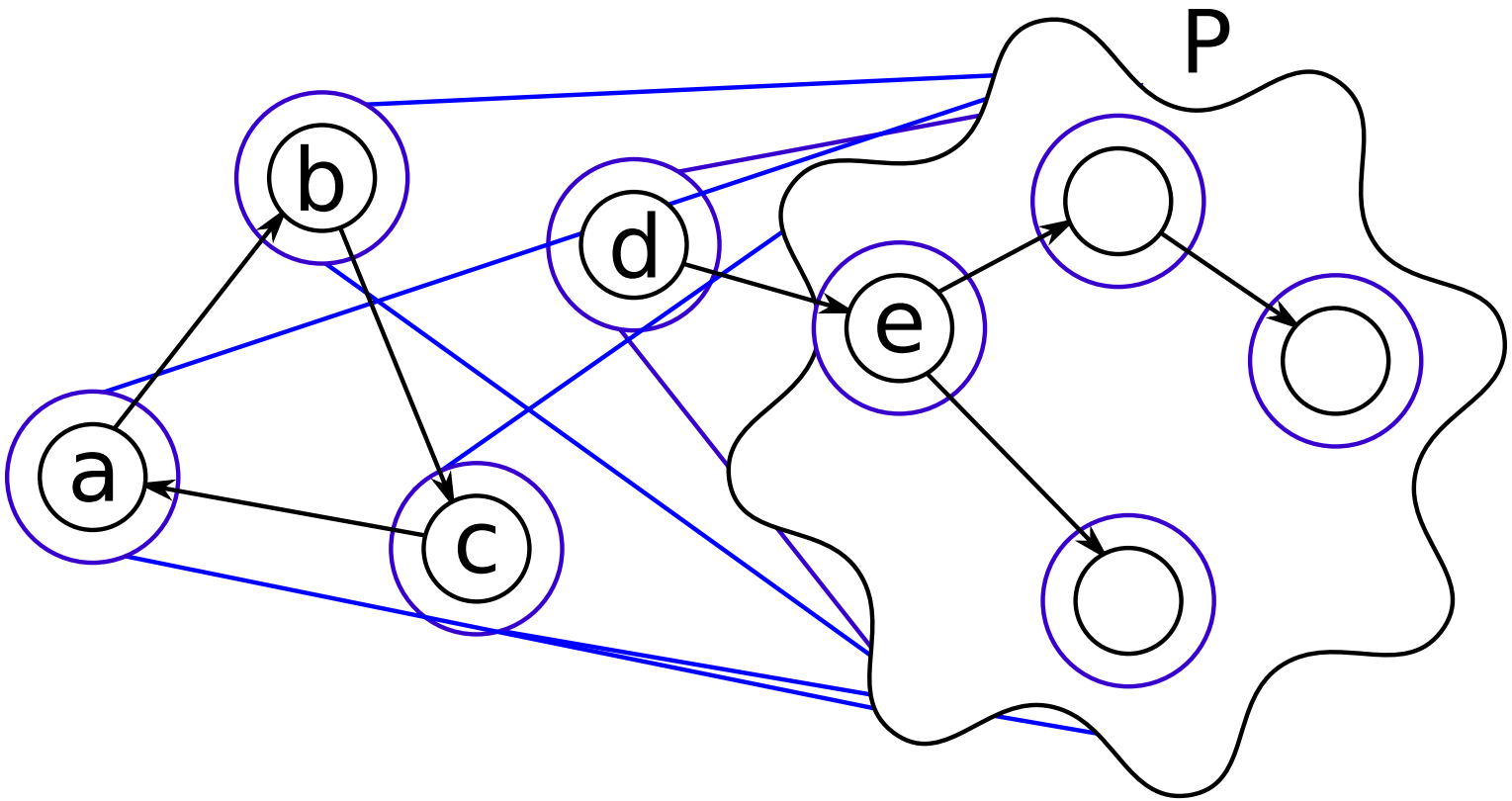
Proving with multiple steps

Closed: $\mu C.R \cup step[C]$



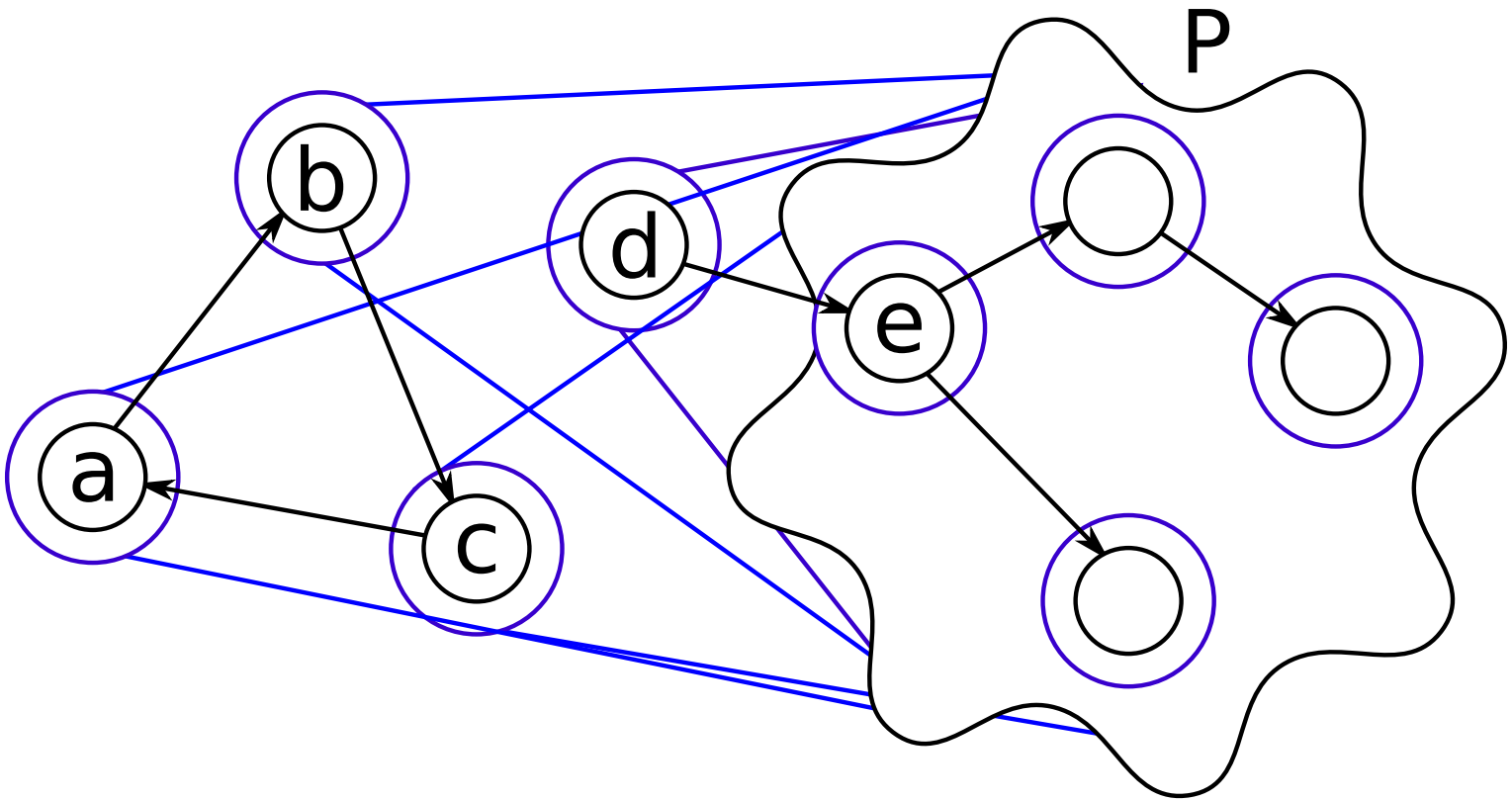
Proving with multiple steps

Final step: $step[\mu C.R \cup step[C]]$



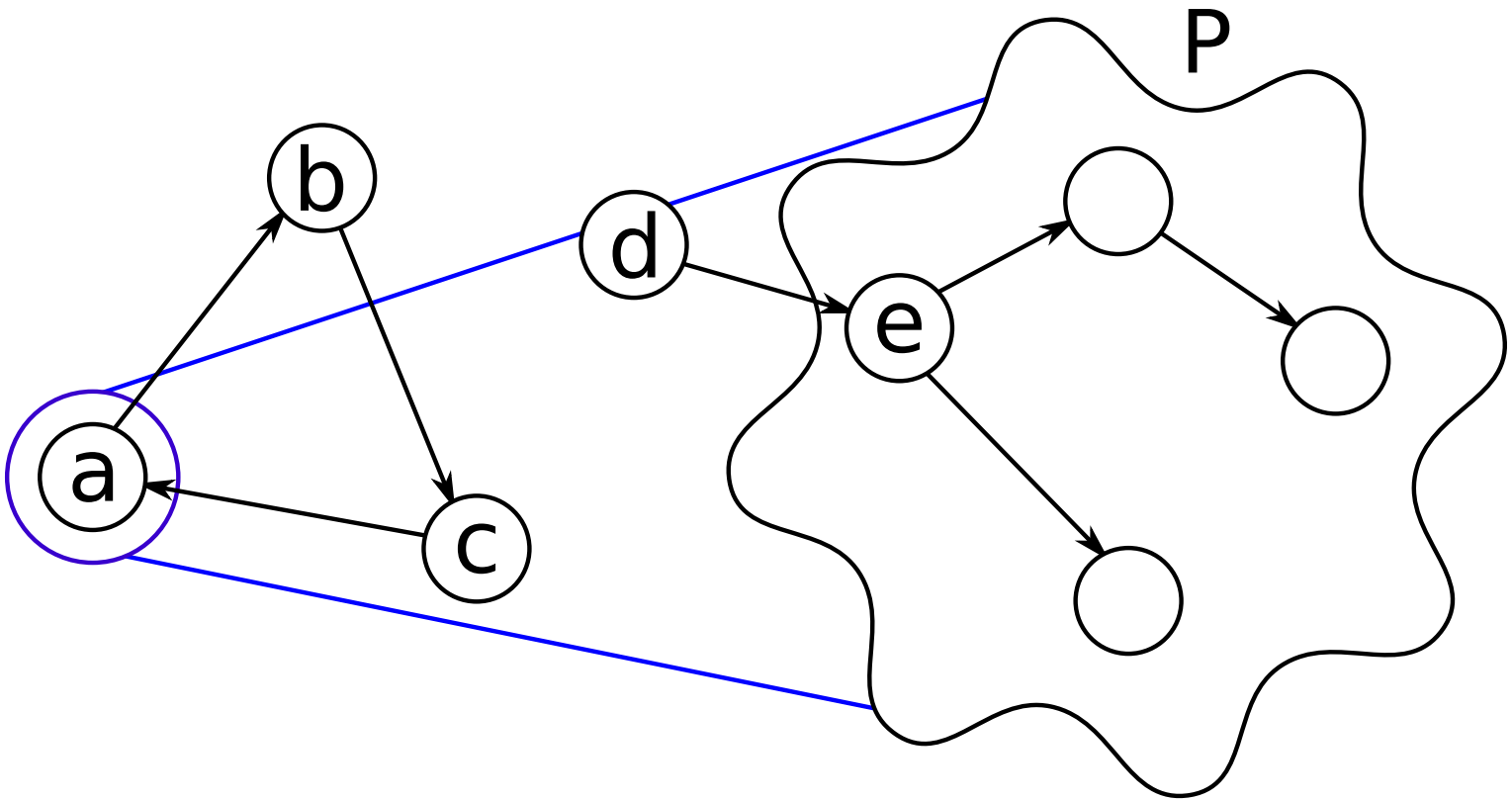
Proving with multiple steps

Supported: $R \subseteq step[\mu C.R \cup step[C]]$



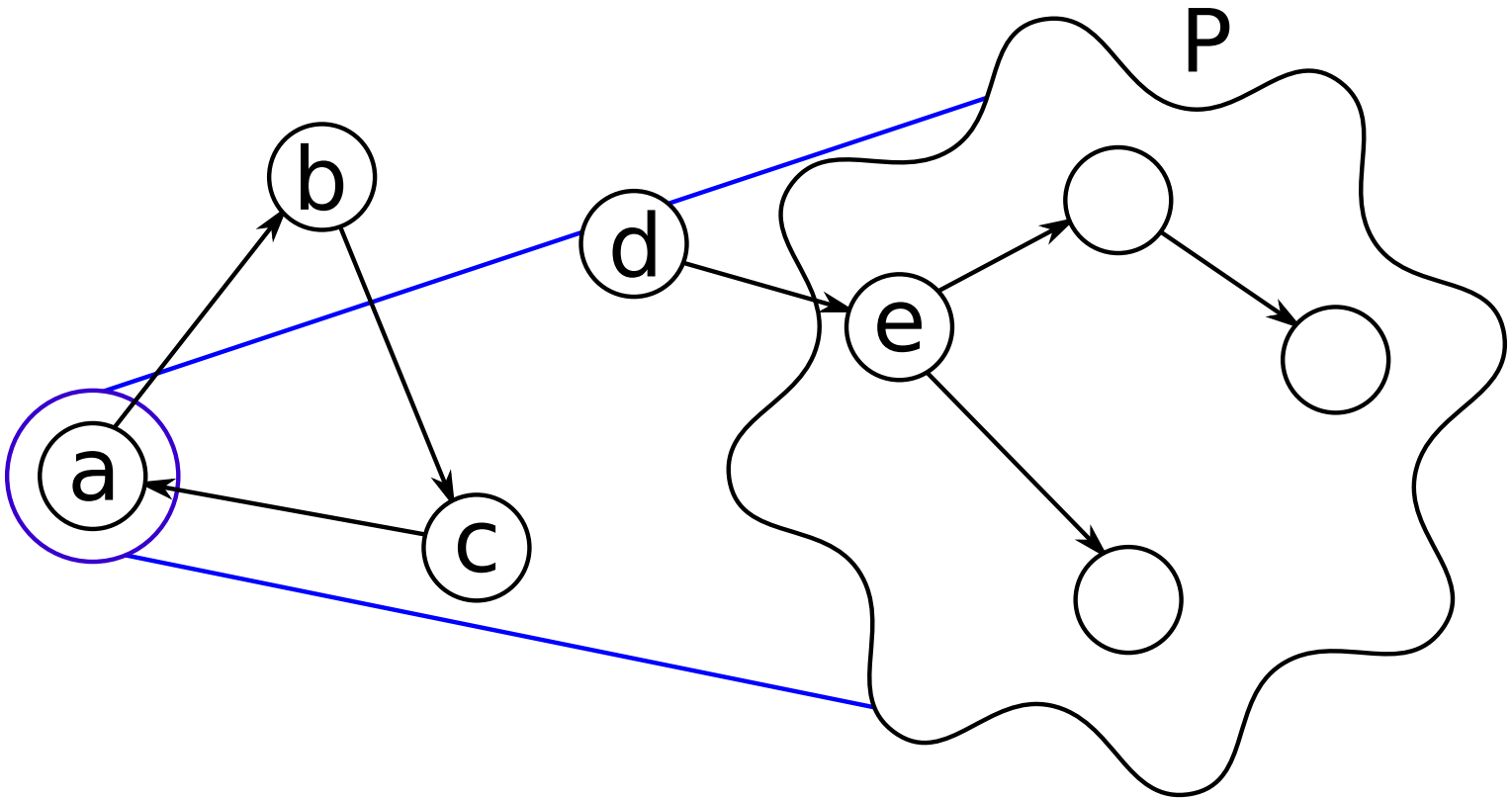
Reasoning forward

Goal: any R which includes (a, P)



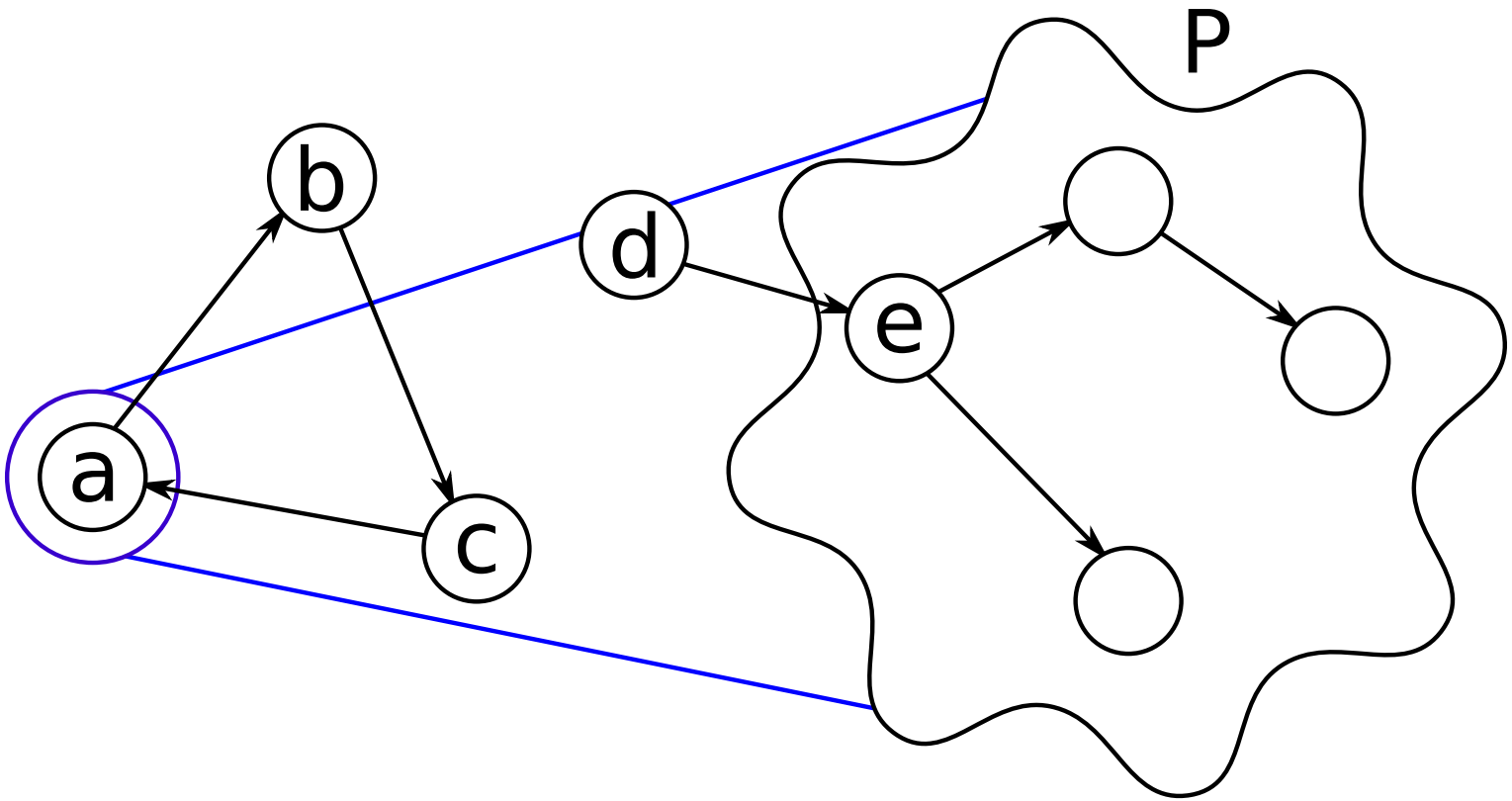
Reasoning forward

$(a, P) \in step[\mu C.R \cup step[C]]$



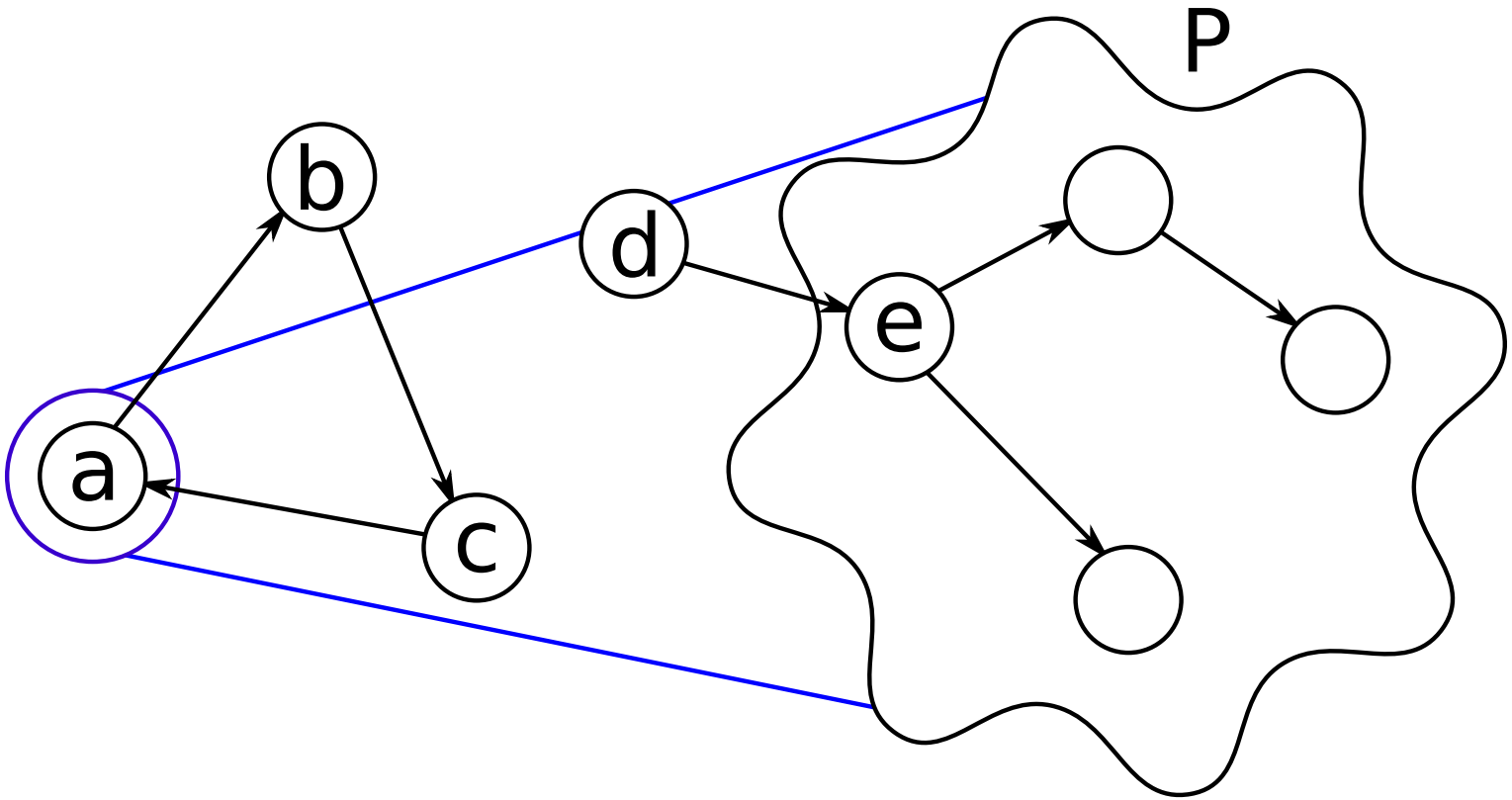
Reasoning forward

$(a, P) \in \mathit{done} \cup \mathit{next}[\mu C.R \cup \mathit{step}[C]]$



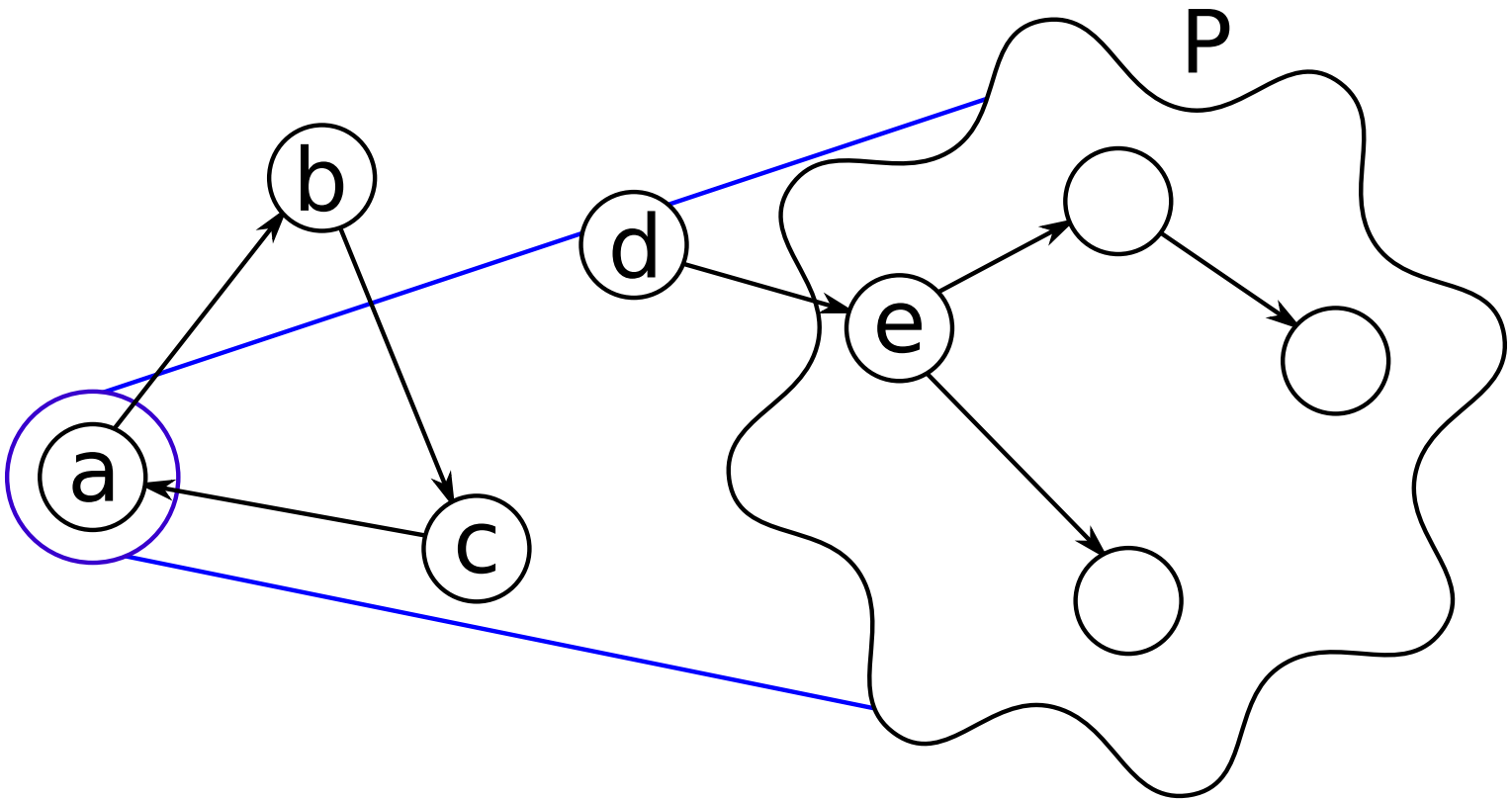
Reasoning forward

$(a, P) \in next[\mu C.R \cup step[C]]$



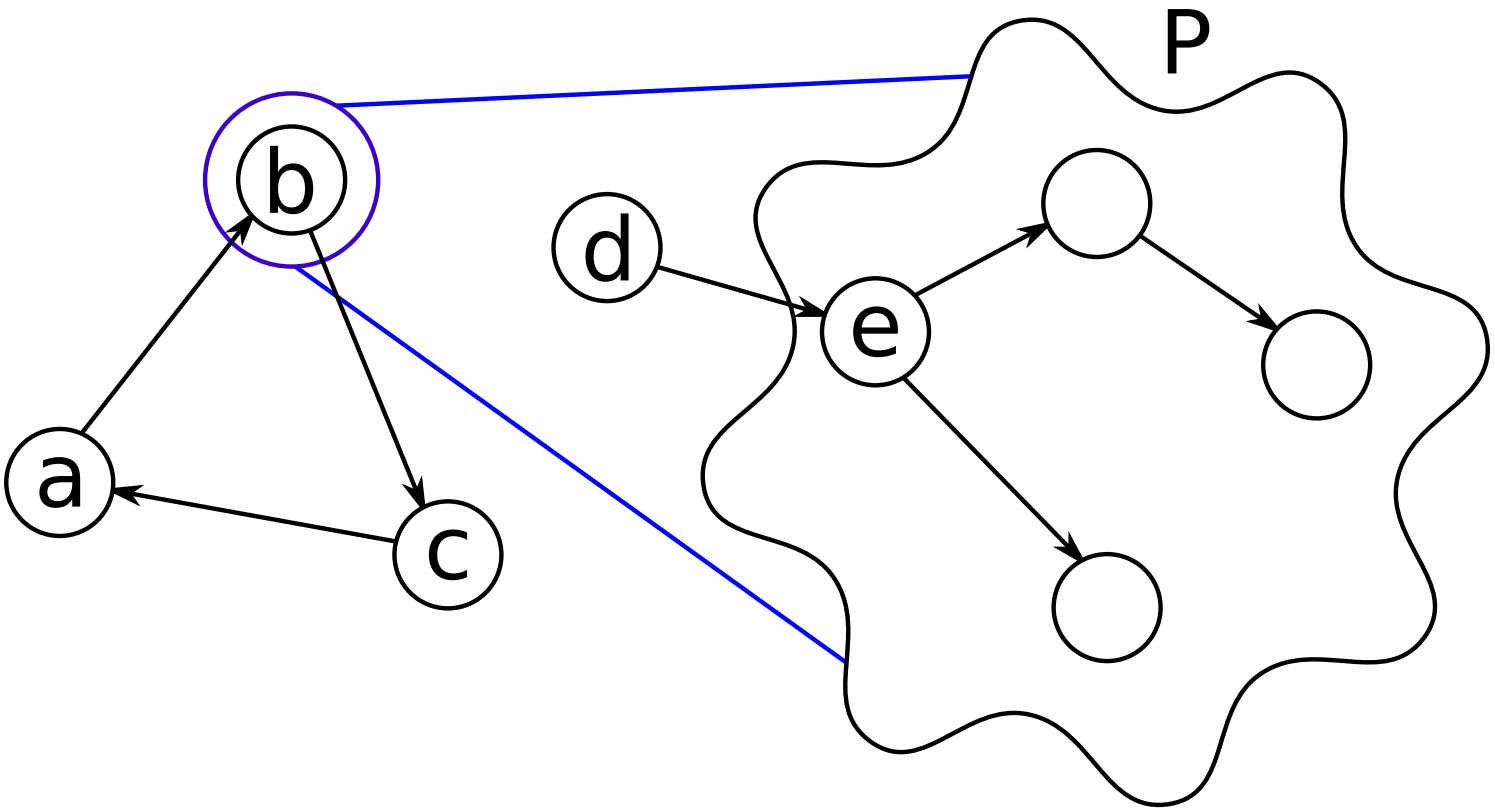
Reasoning forward

$(b, P) \in \mu C.R \cup step[C] \wedge a\ S\ b \rightarrow (a, P) \in next[\mu C.R \cup step[C]]$



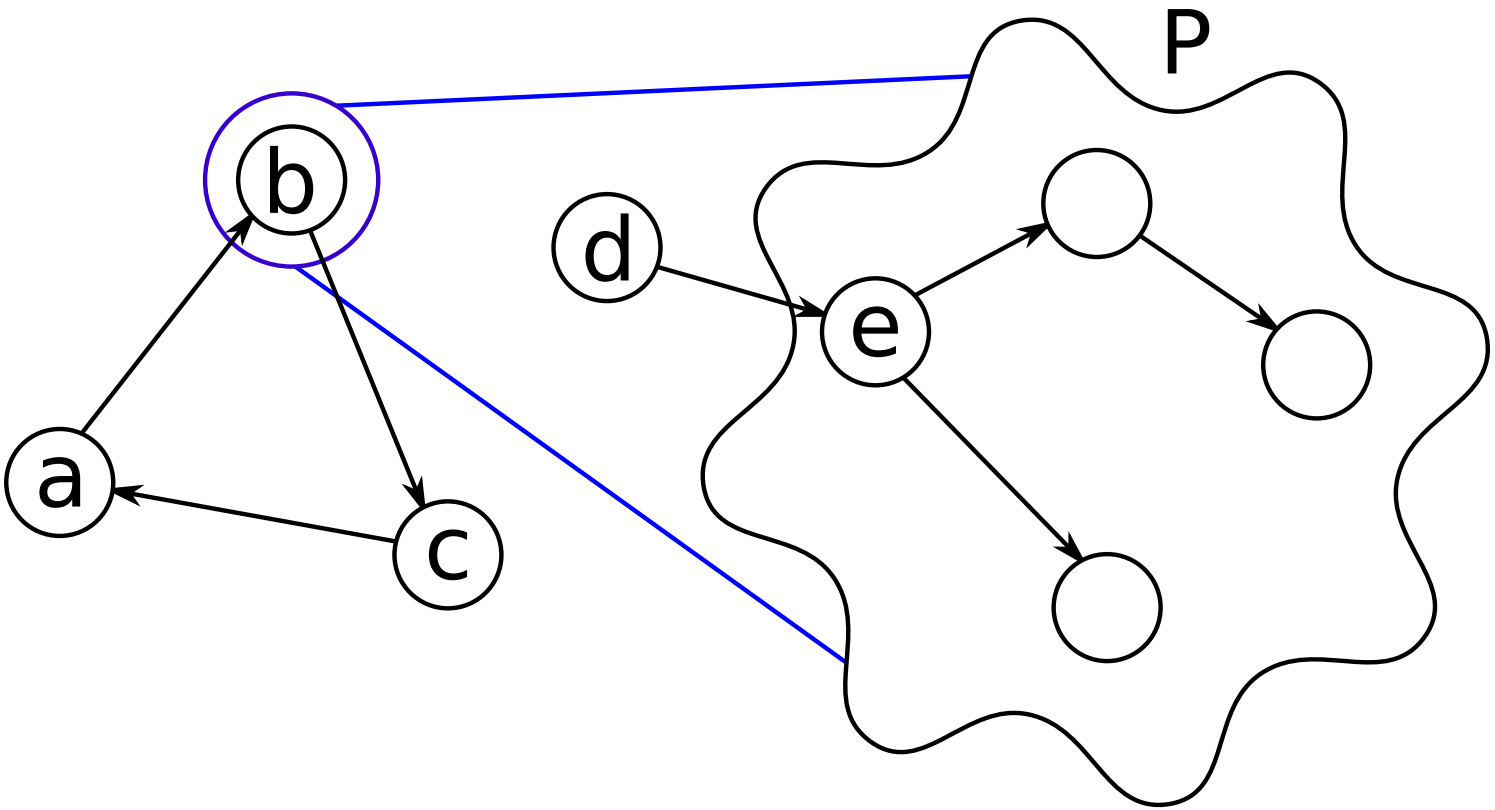
Reasoning forward

$(b, P) \in \mu C.R \cup step[C]$



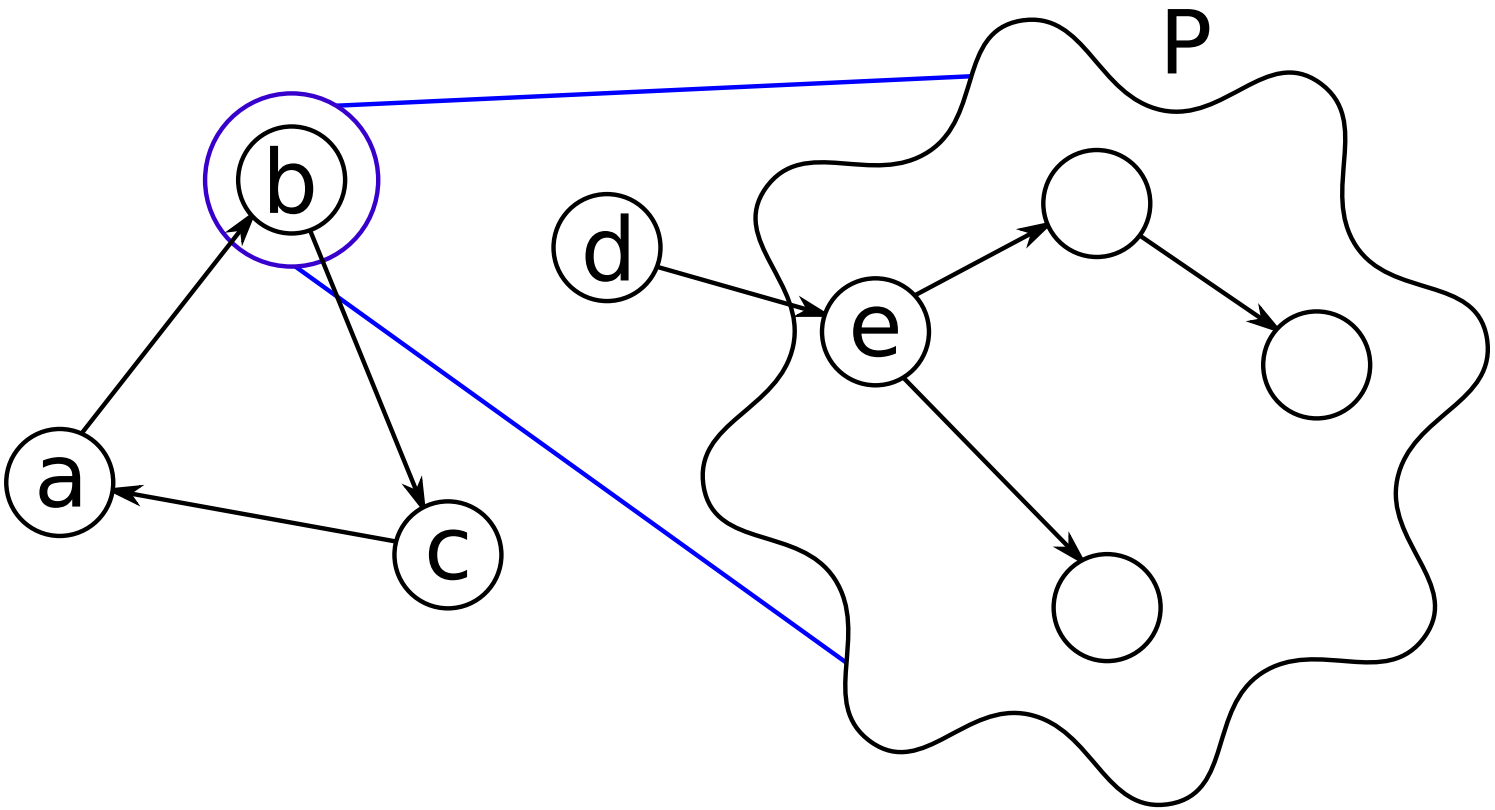
Reasoning forward

$(b, P) \in R \cup step[\mu C.R \cup step[C]]$



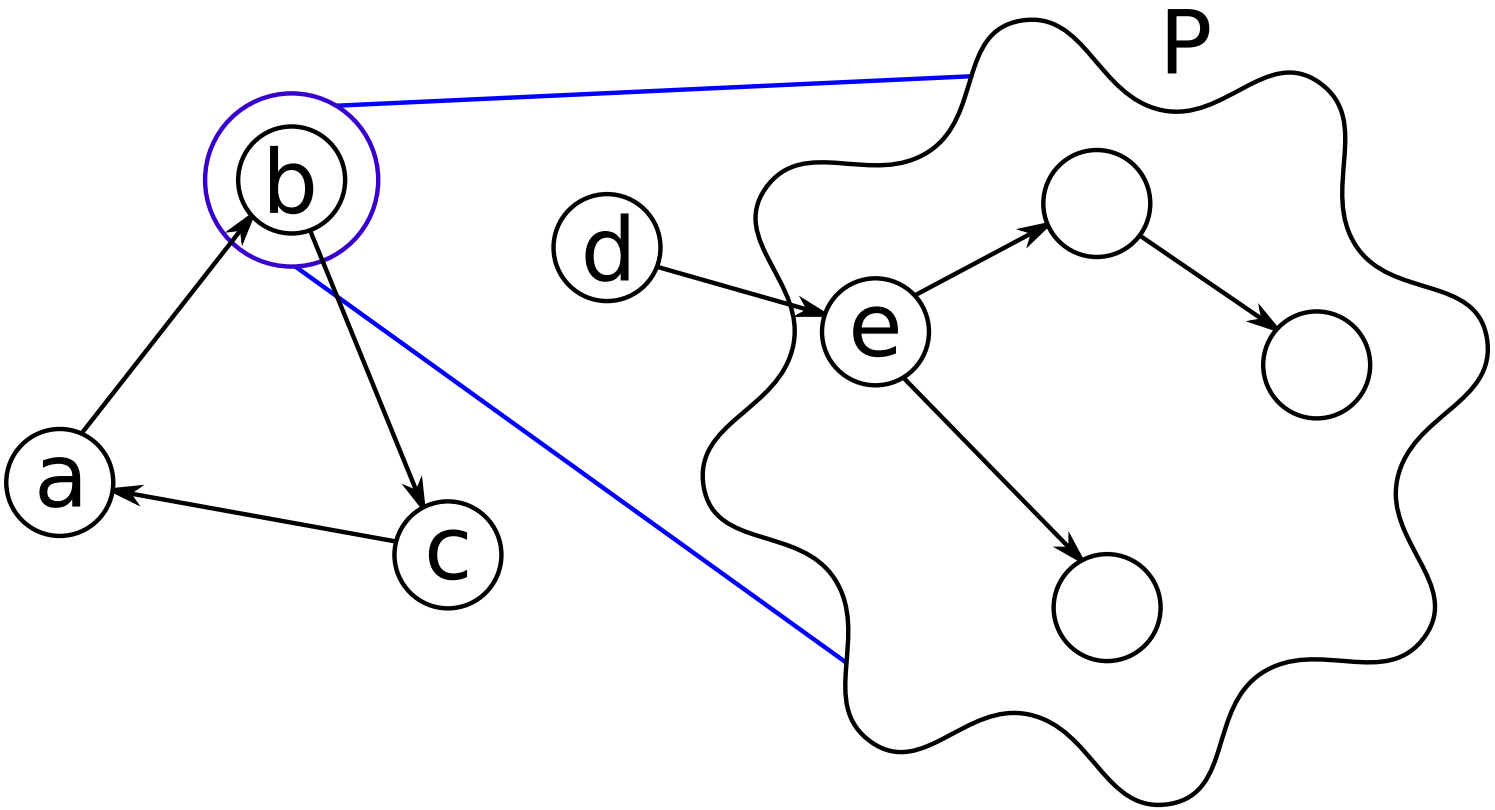
Reasoning forward

$(b, P) \in step[\mu C.R \cup step[C]]$



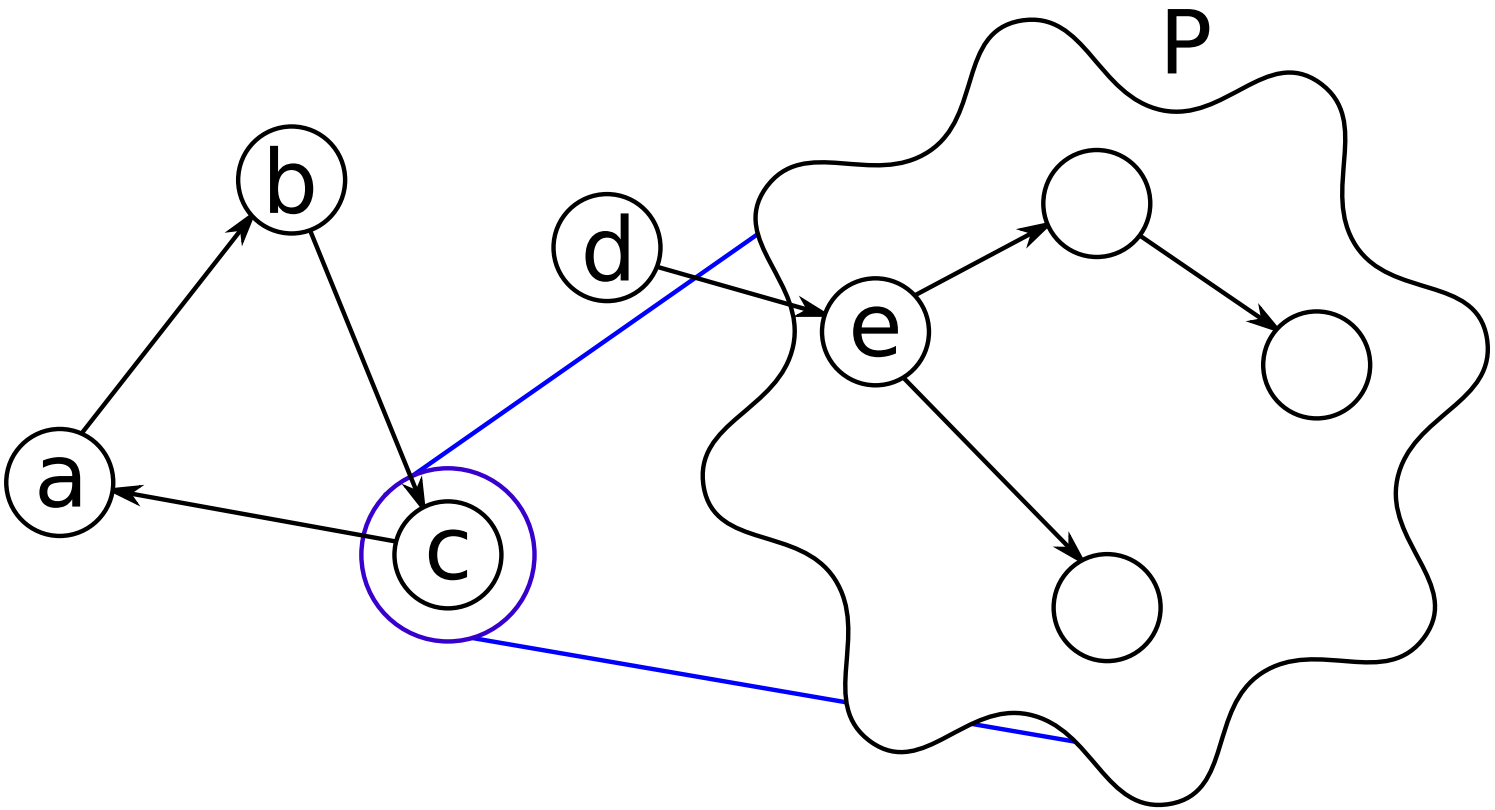
Reasoning forward

$(b, P) \in next[\mu C.R \cup step[C]]$



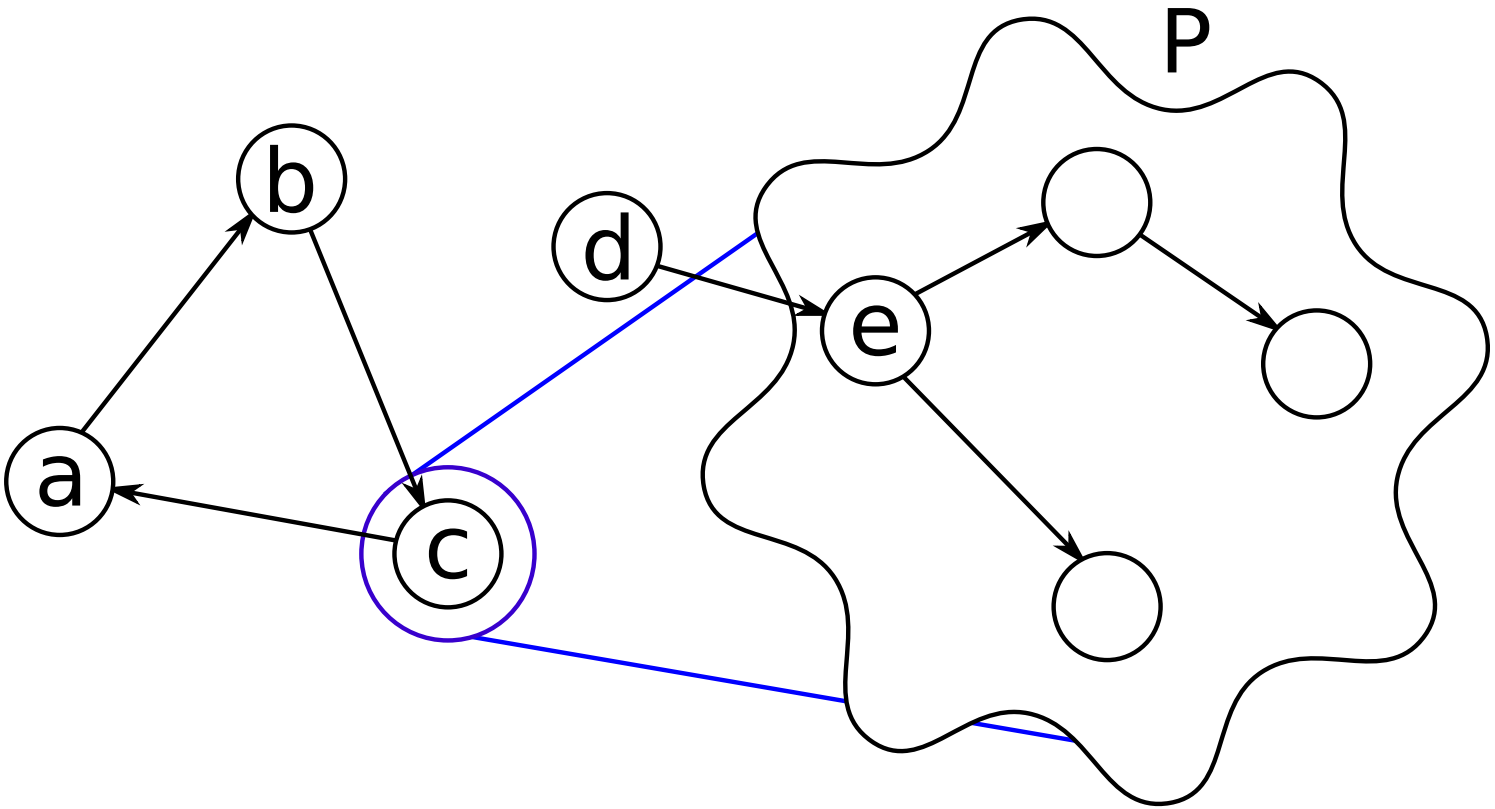
Reasoning forward

$(c, P) \in \mu C.R \cup step[C]$



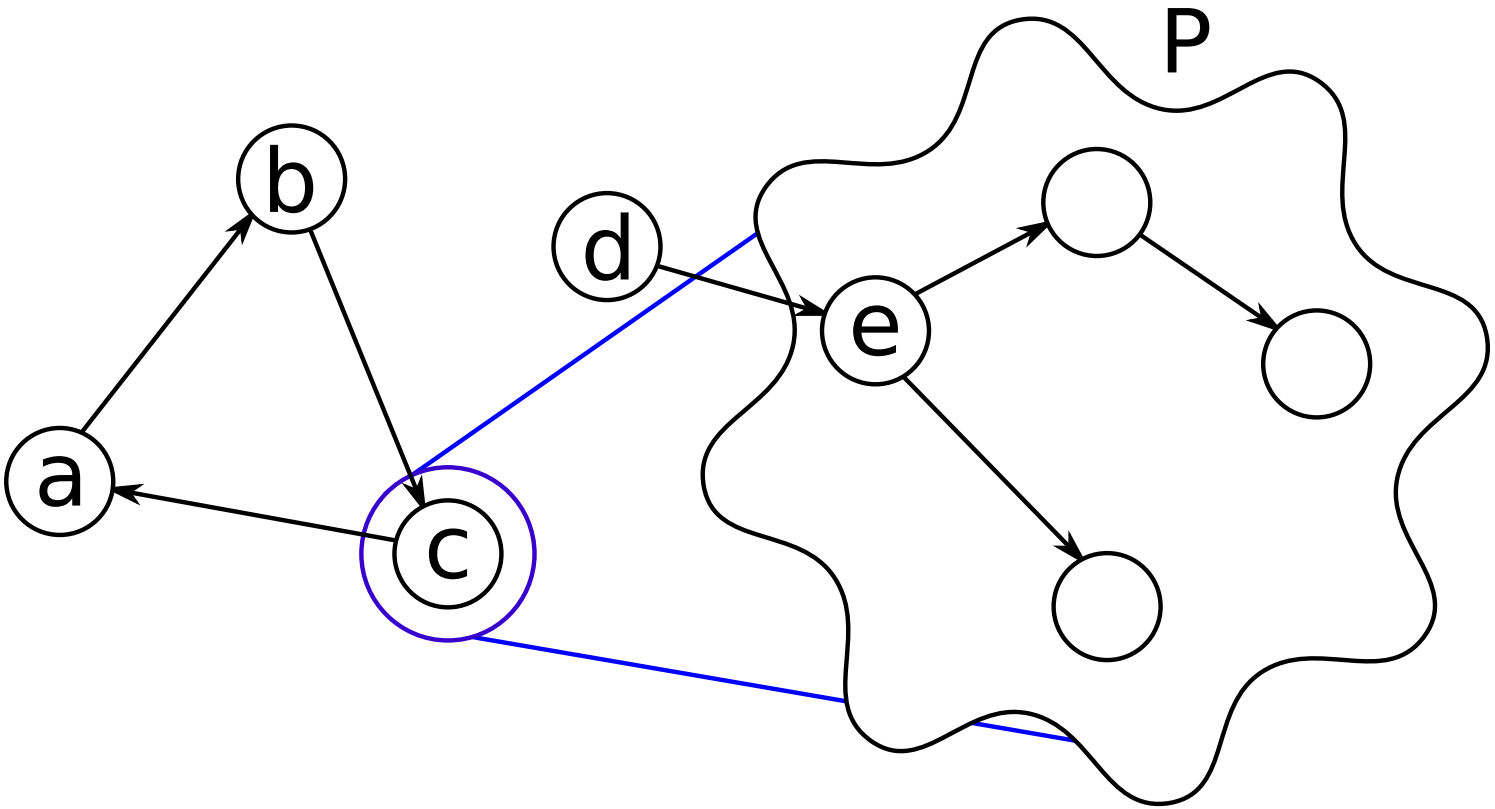
Reasoning forward

$(c, P) \in R \cup step[\mu C.R \cup step[C]]$



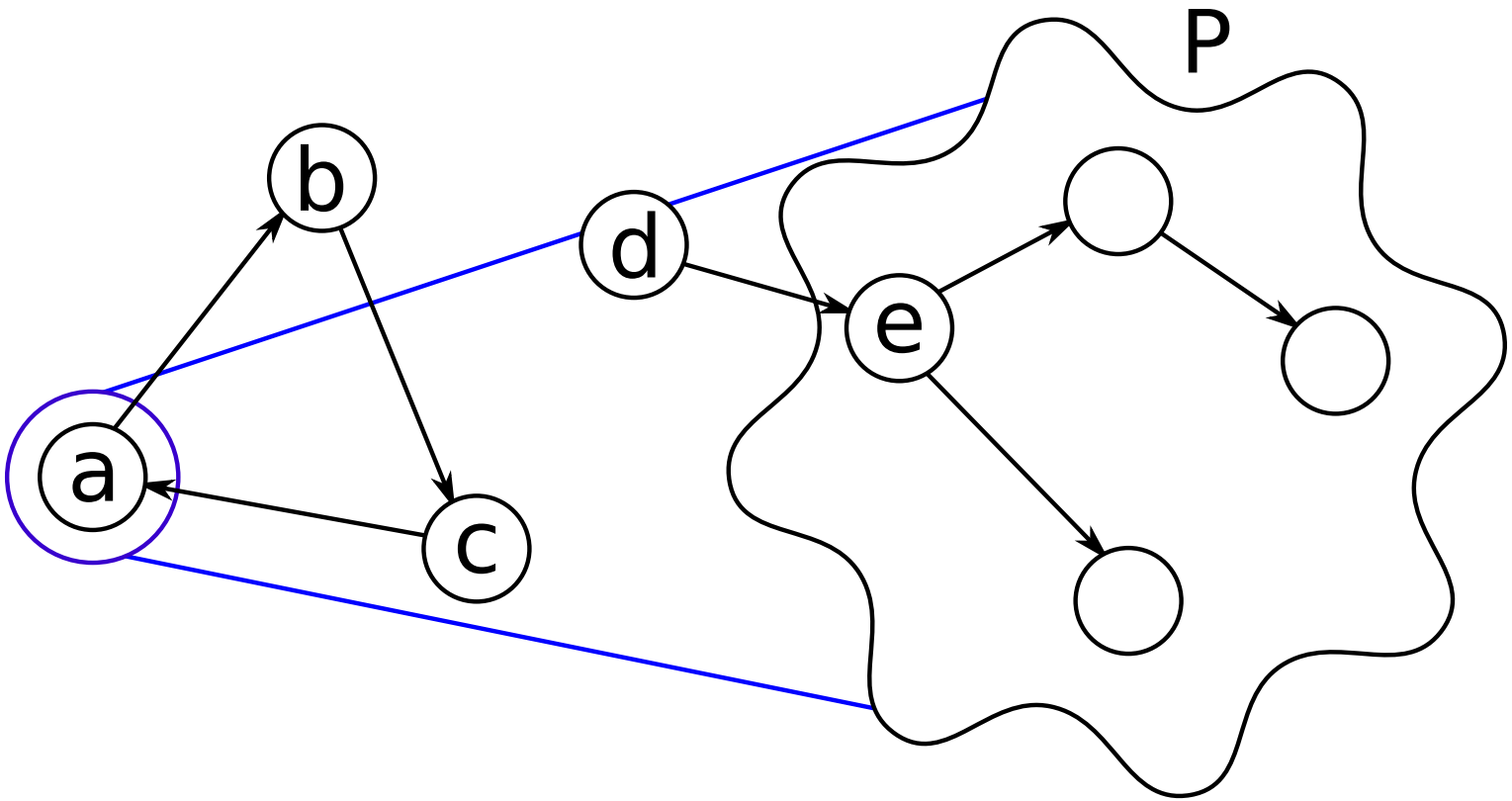
Reasoning forward

$(c, P) \in next[\mu C.R \cup step[C]]$



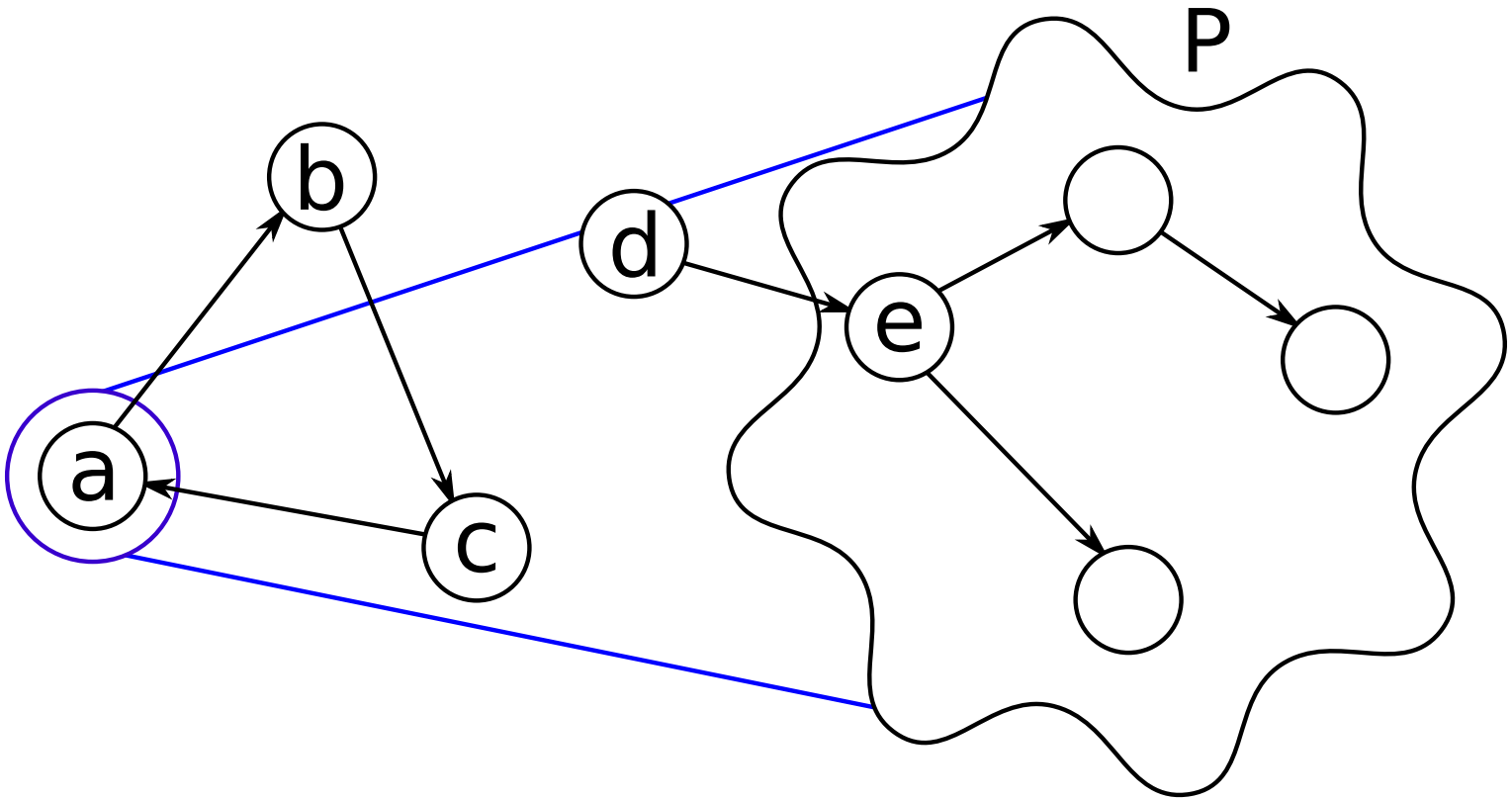
Reasoning forward

$(a, P) \in \mu C.R \cup step[C]$



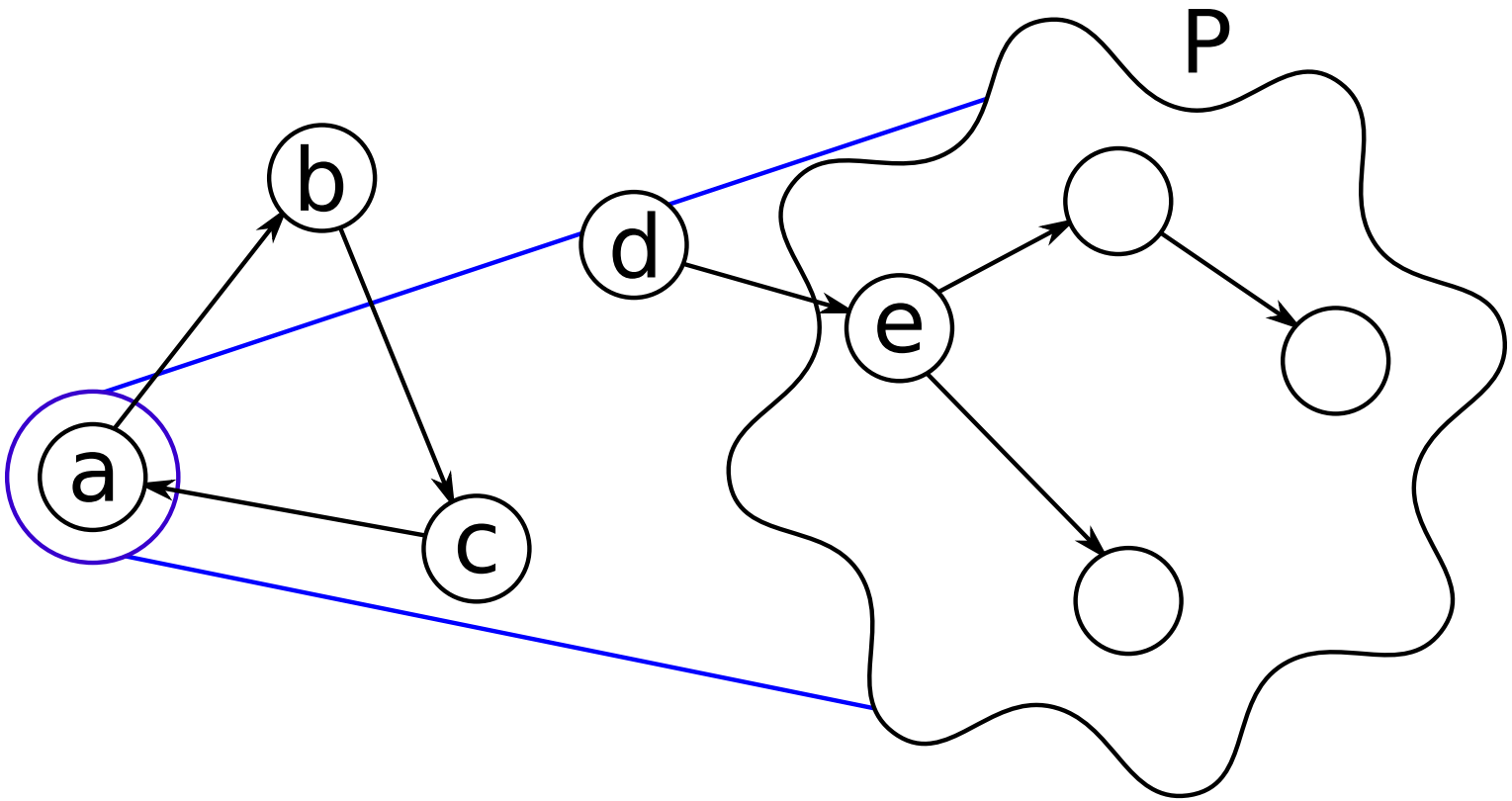
Reasoning forward

$(a, P) \in R \cup step[\mu C.R \cup step[C]]$



Reasoning forward

$(a, P) \in R$



Multi-step coinduction on a program

The set of claims

$\langle \texttt{while(n!=0) \{n=n-1\}}, \{\texttt{n} \mapsto n_0\} \rangle \Rightarrow \{\langle \texttt{skip}, \{\texttt{n} \mapsto 0\} \rangle\}$

is *step*-stable with multiple sets.

Transitivity

Want $x \Rightarrow Q$ if $x \Rightarrow P$ and $y \Rightarrow Q$ for all $y \in P$.
As a function $trans : \mathcal{P}(claims) \rightarrow \mathcal{P}(claims)$

$$trans[X] = \{(x, Q) \mid \exists P.(x, P) \in X \wedge \forall y \in P.(y, Q) \in X\}$$

As before, given $R : \mathcal{P}(claims)$, close under *step* and *trans* by fixpoint

$$\mu C.R \cup step[C] \cup trans[C]$$

Definition

$R : \mathcal{P}(claims)$ is *step-stable using multiple steps and transitivity* if

$$R \subseteq step[\mu C.R \cup step[C] \cup trans[C]]$$

Is this sound?

General rules

In general, any monotone $F : \mathcal{P}(claims) \to \mathcal{P}(claims))$ may be rules

Definition

Given monotone $F : \mathcal{P}(claims) \to \mathcal{P}(claims)$ and claims $R : \mathcal{P}(claims)$, define the closure
$derived : (\mathcal{P}(claims) \to \mathcal{P}(claims)) \times \mathcal{P}(claims) \to \mathcal{P}(claims)$

$$derived[F, R] = \mu D.R \cup step[D] \cup F[D]$$

Proving with rules

Definition

For monotone $F : \mathcal{P}(claims) \to \mathcal{P}(claims))$, a set $R : \mathcal{P}(claims)$ is *step-stable using* F if

$$R \subseteq step[derived[F, R]]$$

Rule Validity

Definition

Monotone $F : \mathcal{P}(claims) \to \mathcal{P}(claims)$ is *simply valid* if for any $X : \mathcal{P}(claims)$,

$$X \subseteq step[derived[F, X]]$$

implies

$$derived[F, X] \subseteq step[derived[F, X]]$$

Compositional Validity

Composition

Given functions $F, G : \mathcal{P}(claims) \to \mathcal{P}(claims)$, define *composition* by

$$(F + G)[X] = F[X] \cup G[X]$$

Composition preserves monotonicity, but not simple validity: strengthen

Definition

$F : \mathcal{P}(claims) \to \mathcal{P}(claims)$ is *compositionally valid* if it is monotone and

$$\forall X. F[step[X]] \subseteq step[derived[F, X]]$$

Validity and Compositionality of Compositional Validity

Validity Lemma

If $F : \mathcal{P}(claims) \to \mathcal{P}(claims)$ is compositionally valid and $R \subseteq claims$ satisfies

$$R \subseteq step[derived[F, R]]$$

then

$$R \subseteq derived[F, R] \subseteq step[derived[F, R]] \subseteq reaches$$

Compositionality Lemma

If $F, G : \mathcal{P}(claims) \to \mathcal{P}(claims)$ are both compositionally, then $(F + G)$ is also compositionally valid

Validity Example: *trans*

To show *trans* is compositionally valid, fix $R \subseteq \mathit{claims}$, assume

$$(x, P) \in \mathit{step}[R] \quad \mathit{forally} \in P.(y, Q) \in \mathit{step}[R]$$

- If $x \in P$ then $(x, Q) \in \mathit{step}[R]$ by second hypopthesis
- Else $x\, S\, z$ with $(z, P) \in R$. Then $(z, Q) \in \mathit{trans}[R \cup \mathit{step}[R]]$, $(x, Q) \in \mathit{step}[\mathit{trans}[R \cup \mathit{step}[R]]]$.
- In either case,

$$(x, Q) \in \mathit{step}[\mathit{derived}[\mathit{trans}, X]]$$

Interlude: Combining Proofs

To show

$$(prog \cup library \cup internals) \subseteq step[derived[(trans + weaken + F), (prog \cup library \cup internals)]]$$

it suffices to show

$$prog \subseteq step[derived[trans + F], prog \cup library]]$$

and

$$library \cup internals \subseteq step[derived[trans + weaken, library \cup internals]]$$

"Second-Order" Program

What about higher order programs? Consider

```
add3(plus,x,y,z) = plus(plus(x,y),z)
```

Specified by

$$\forall p.\quad (\forall a\ b\ E\ \sigma.\langle E[p(a,b)],\sigma\rangle \Rightarrow \{\langle E[a+b],\sigma\rangle)\} \rightarrow$$
$$\forall x\ y\ z\ E\ \sigma.\langle E[\mathsf{add3}(p,x,y,z)],\sigma\rangle \Rightarrow \{\langle E[x+y+z],\sigma\rangle\}$$

"Second-Order" Specification

First-order specifications became sets of claims. Now abstract over set of claims to consider true in premises. Specification becomes function $Spec : \mathcal{P}(claims) \to \mathcal{P}(claims)$

$$Spec[H] = \{(\langle E[\texttt{add3}(p, x, y, z)], \sigma\rangle, \{\langle E[x + y + z], \sigma\rangle\}) \mid \forall a\ b\ E'\ \sigma'.(\langle E'[p(a, b)], \sigma'\rangle, \{\langle E'[a + b)], \sigma'\rangle)\}) \in H\}$$

"Second-Order" Proof

Functions from claims to claims used as derived rules, re-use validity condition.
If F is compositionally valid,

$$\forall X.\, Spec[step[X]] \subseteq step[derived[F + Spec, X]]$$

implies $F + Spec$ is compositionally valid

Lemma

If $F + Spec$ and F are compositionally valid,

$$Spec[reaches] \subseteq reaches$$

Higher-Order properties

In general, may take and return higher-order functions.
Simple types

$$\tau := \mathbb{N} \mid \tau \to \tau$$

give a reachability property

$$\begin{aligned} \mathit{type}(\mathbb{N}, v) &= v \text{ is a number} \\ \mathit{type}(A \to B, v) &= \forall a.\ \mathit{type}(A, a) \to \\ &\qquad \langle E[v(a)] \rangle \Rightarrow \{\langle E[r] \rangle \mid \mathit{type}(B, r)\} \end{aligned}$$

Can't translate to spec of form $\mathcal{P}(\mathit{claims}) \to \mathcal{P}(\mathit{claims})$, monotonicity ruined by positive and negative occurances of $\Rightarrow$

Mixed-Variance Predicates

Splitting argument suffices:

$$type : \mathcal{P}(claims) \to \mathcal{P}(claims) \to bool$$

$$\begin{aligned} type(\mathbb{N}, \quad v, Neg, Pos) &= v \text{ is a scalar} \\ type(A \to B, v, Neg, Pos) &= \forall a.type(A, a, Pos, Neg) \to \\ & (\langle E[v(a)] \rangle, \{\langle E[r] \rangle \mid type(B, r, Neg, Pos)\}) \in Pos \end{aligned}$$

Is monotone in *Pos*, anti-monotone in *Neg*

Higher-Order proof

A higher-order specification is function

$$Spec : \mathcal{P}(claims) \to \mathcal{P}(claims) \to \mathcal{P}(claims)$$

which is anti-monotone in the first argument and monotone in the second argument.

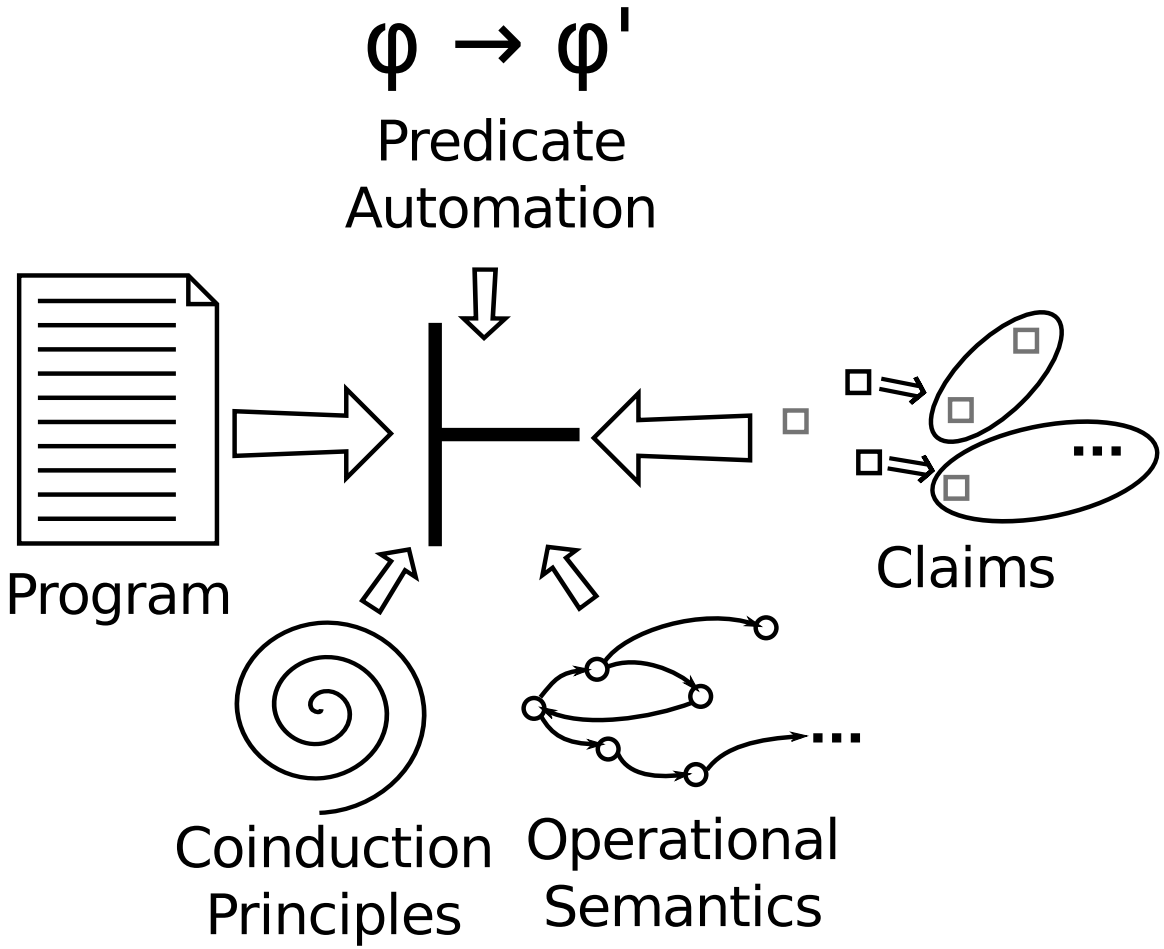
Proved under rules F when

$$\begin{aligned}
&\forall(Neg\ \ Pos : \mathcal{P}(claims)) \ . \\
&\quad step[derived[F, Spec[Neg, Pos]]] \subseteq Neg \to \\
&\quad Pos \subseteq step[derived[F, Spec[Neg, Pos]]] \to \\
&\quad\quad Spec[Neg, Pos] \subseteq step[Spec[Neg, Pos]]
\end{aligned}$$

When $Spec$ proved under F and rules F valid, there exist some $N, P \subseteq claims$ such that

$$Spec[N, P] \subseteq reaches$$

Project Structure



Coinduction Principles

- Formalize in Coq
- Publish Basic Theory
  - Submit to LICS 2014, mid January, with MatchC examples.
- Analyze derived rule conditions
- Develop higher-order case

Operational Semantics

- Handwritten semantics to start
- Generate from K definitions
  - Reuse predicates, semantic domains from handwritten semantics.
- Try existing Coq semantics, e.g. OTT

Operational Semantics - Claims

Also need claims

- Begin writing by hand
- Extend K translator for annotations
- Abbreviations as needed
  - Filling code component by embedding annotations in program
  - Hoare style variable lookup abbreviation?
  - Automatic threading of mixed-variance predicates?

Automating Verification

Two major sorts of automation

- Order for applying proof rules
  - MatchC successful with simple tactic: try in order to
    1. Finish
    2. Apply claim from specification
    3. Take a step in semantics
    4. Make a case split to enable above
       (generate tactic by analyzing semantics?)
  - May replay trace from K reachability prover.
- Folding, unfolding, simplifying predicates of specification
  - Pure single-state domain reasoning, should be able to borrow

Validating Power

Prove properties handled by

- MatchC (trees, Schorr-Waite)
- Bedrock (allocator, cooperative threads)
- FLINT (self-modifying code, interrupts)

Validating Automation

After power, look at proof effort

- Bedrock
- Boogie
- Why

Validating Adequacy

Proofs sound with respect to Coq semantics, may not match K interpreter

- Match exection of K tests
- Generate executable semantics, extract interpreter
- Formalize K, translate by deep embedding

End

Reachability is Greatest Fixpoint

fixed

If x diverges, it has an immediate successor y which also diverges. If $x\, S^*\, z$ for some $z \in P$, then either $x \in P$ already or x has an immediate successor y with $y\, S^*\, z$ as well, so $\mathit{reaches} = \mathit{step}[\mathit{reaches}]$.

greatest

Suppose $X \subseteq \mathit{step}[X]$, and (x, P) is a claim in X. Consider whether x diverges. If so, $(x, P) \in \mathit{reaches}$. If x terminates, S is well-founded under x, so by induction assume any claim (y, P) in X with $x\, S^+\, y$ is also in $\mathit{reaches}$. As X is step-stable, either $x \in P$ already, or x has some successor y with $(y, P) \in X$. In either case, $(x, P) \in \mathit{reaches}$.

Back

ω not always enough

The least fixpoint is not necessarily reached by iterating a countable number of times from $\emptyset$. The set of states which terminate along all paths is the least fixpoint of

$$F[S] = \{x \mid \forall y.x\ S\ y \rightarrow y \in S\}$$

but

$$\cup_{i=0}^{\infty} F^i[\emptyset]$$

misses programs such as the one that picks any natural number, and then runs for that many more steps. It's found in the next step:

$$F[\cup_{i=0}^{\infty} F^i[\emptyset]]$$

i.e, transfinite induction up to $\omega + 1$. Higher ordinals can be required.

Back